版权注意事项：
1、书籍版权归作者和出版社所有
2、本PDF仅限用于个人获取知识，进行私底下的知识交流
3、PDF获得者不得在互联网上以任何目的进行传播
4、如觉得书籍内容很赞，请购买正版实体书，支持作者
5、请于下载PDF后24小时内删除本PDF。

HZ BOOKS
华章科技

石桥码农 编著

小程序从0到1

微信全栈工程师一本通

从前端到后端全面讲解小程序开发所需的所有技术
实例讲解加线上辅导，在实践中构建知识体系

由浅入深，循序渐进，
让新手轻松入门

机械工业出版社
China Machine Press

内容简介

本书全面介绍了小程序需要用到的所有技术，从小程序组件到 WXSS 样式，从前端 JS 语言到后端 Go 语言，并通过实战案例，由浅入深，一步一步引领读者学习小程序开发技巧。全书共分 4 篇：

第一篇（第 1 章），介绍开发环境的准备及账号的注册，完成第 1 章的学习相当于取得了步入小程序技术殿堂的入场券。

第二篇（第 2~10 章），是项目实战部分，其中第 2~6 章是小程序前端案例，使用了后台接口但未涉及后台编程；第 7~10 章则在已有案例的基础上，添加了后端程序的支持。先学习前端，再学习后端，每次专注一个点学习，更易理解和掌握。

第三篇（第 11~14 章），详细地讲解了所有小程序组件的使用方法，所附示例几乎全部可用于生产，这大大降低了初学者在美工上的学习门槛。

第四篇（第 15~17 章），是综合练习部分，系统地介绍了 JS 语言、Go 语言、WXSS 样式语法等必备知识与技能。这 3 章既讲解了工具，可备开发查询之需，又在每节提供了独立的练习代码，便于读者利用碎片时间提高编码水平。

小程序从0到1

微信全栈工程师一本通

石桥码农 编著

机械工业出版社
China Machine Press

图书在版编目（CIP）数据

小程序从0到1：微信全栈工程师一本通/石桥码农编著．—北京：机械工业出版社，2017.12

ISBN 978-7-111-58404-9

I. 小… II. 石… III. 移动终端－应用程序－程序设计 IV. TN929.53

中国版本图书馆CIP数据核字（2017）第268482号

本书分为四篇：第一篇即为第1章，介绍开发环境的准备及账号的注册。学完本章内容，相当于取得了入场券。第二篇包括第2～10章，是项目实战部分。其中第2～6章是小程序前端案例，使用了后台地址但未涉及后台编程；第7～10章是在已有案例的基础上添加了后端程序的支持。先学习前端，再学习后端，由浅入深，这样更易于理解和掌握。第三篇包括第11～14章，详细讲解所有小程序组件的使用方法，几乎全部组件都具有生产可用的UI，所附示例源码可以直接用于实际项目中，大大降低了初学者开发第一个小程序时在美工上的门槛。第四篇包括第15～17章，是综合练习部分，系统地介绍JS语言、Go语言、WXSS样式语法等必备知识与技能。其中各节有独立的练习代码，可供读者实际练习之用。这部分也可作为工具文档，供开发者参考使用。

小程序从0到1：微信全栈工程师一本通

出版发行：机械工业出版社（北京市西城区百万庄大街22号　邮政编码：100037）
责任编辑：杨绣国　　责任校对：殷　虹
印　　刷：中国电影出版社印刷厂　　版　　次：2018年1月第1版第1次印刷
开　　本：186mm×240mm　1/16　　印　　张：15.75
书　　号：ISBN 978-7-111-58404-9　　定　　价：59.00元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88379426　88361066　　投稿热线：（010）88379604
购书热线：（010）68326294　88379649　68995259　　读者信箱：hzit@hzbook.com

版权所有·侵权必究
封底无防伪标均为盗版
本书法律顾问：北京大成律师事务所　韩光/邹晓东

Forword 推荐序

小程序是一个能让微信更加好玩、更具互动性的新技术。不久前，我所在的一个微信群，大家表达了选出一个“群主管理会”的需求。一位老同学花了一点时间用小程序做了一款“投票神器”的小工具，几百位老同学都参与了投票，并盛赞该工具的方便好用。

这也是我第一次使用小程序的体验，印象非常深刻。小程序开发的产品不用安装，打开即用，使用简单，易于推广。当然，小程序开发的产品不仅仅是简单的投票工具，其应用场景也许只受限于人们的想象力和市场的需求。

在万众创新、万众创业的时代，随着众多商家依托小程序获得了红利，企业对小程序的需求日益高涨。小程序不仅是大学毕业生以及职场人就业谋生的技能，也是中小企业低成本开发产品、快速进行市场验证的首选技术方案。

本书的作者石桥码农，曾是我的学生，也是工作上的小伙伴。他作为一名程序员，在艺术方面也有相当的追求和实践，虽自称码农，其实是把写程序当成一门艺术对待。他对程序代码的要求一向一丝不苟、精益求精。我至今记得他形容糟糕的代码为“穿着棉袄洗澡”，十分形象生动。他的技术文档一向秩序井然，不堆砌词藻，不故弄玄虚，注解详尽而不失简约，语言朴素而又富含感染力，因此有理由相信他所编写的这本书一定能对初学者有所裨益。

中国传媒大学脑科学与智能媒体研究院院长
曹立宏教授

前 言 *Preface*

为什么要写这本书

2017 年 4 月 22 日，我在知乎发起了一场“零基础周末学习小程序开发”直播，从当晚 8 点开始，我一边撰写教程笔记，一边与 500 多位学员在线互动。教学从注册账号开始，接着是下载微信开发者工具，然后创建第一个 quick start 项目，最后编写后端代码，并在微信上运行和测试所开发的小程序。从那天晚上到第二天凌晨 4 点，我发出了 7 篇教程。

在这场直播中，我原本以为大家会提问一些诸如页面如何跳转、数据如何缓存等技术问题，但是大家提的却大都是一些有关小程序的边缘问题，诸如如何下载和安装小程序、如何获得小程序内测资格等。

不少学员尚不知道小程序已于 2017 年 1 月 9 日正式上线；并且，个人也能注册账号；所谓的 200 个小程序内测资格已经成为过去式了；而且小程序不需要下载安装。

很多学习小程序开发的学员甚至毫无编程基础，他们对如何开发一款小程序一无所知。由此我意识到，小程序初学者最迫切需要的并不是复杂和高深的教程，而是一本简单而全面地介绍小程序开发的图书。全面与快速入门是其第一需求，基于此，笔者编写了本书。

小程序不是一门语言，它是一门新的综合应用技术。小程序无须下载，不用安装，拿来即用，正所谓“事了拂衣去，不留身与名”。凡是接触过原生 iOS、Android 应用开发的读者，都能理解传统开发技术带给开发者的痛苦，如机型繁多、适配困难、审核周期长（iOS 应用），等等。

达尔文说过，“自然界生存下来的，既不是四肢最强壮的，也不是头脑最聪明的，而是有能力适应变化的物种。”

国内 App 的运营成本一直在增长，目前获取一个新用户的成本甚至高达 30 元人民币。在这种环境下，微信的小程序应运而生。从小程序的更新历史来看，微信之父张小龙打造新技术生态圈的决心是异常坚决的。随着小程序技术的成熟，开发者社区的形成，在第一批小程序开发者赚到第一桶金时，这一新技术的火爆才刚刚拉开帷幕。

2017 年 3 月 27 日，微信小程序开放了个人账号申请，从此以后，不是企业也能开发小程序。

2017 年 3 月 28 日，微信小程序开放了蓝牙、卡券、获知访问场景、共享微信通讯录等功能，并支持 JS ES6 新语法。

2017 年 4 月 17 日，微信小程序代码包的大小限制由 1MB 提升到 2MB，开放了第三方平台开发小程序的功能，开放了数据分析接口。

2017 年 4 月 20 日，微信小程序对所有公众号都开放了关联小程序的功能。

2017 年 4 月 25 日，微信小程序开放了公众号推送文章可插入小程序的功能。

2017 年 5 月 19 日，微信小程序可支持蓝牙。

2017 年 6 月 21 日，微信小程序开放了打开另一个小程序的功能。

2017 年 7 月 11 日，微信小程序添加了富文本支持。

……

随着微信小程序不断开放新接口与新功能，小程序的开发社区正在逐渐形成。学习一门新技术最好的契机，正是其方兴未艾之时。无论是初入校园的大一新生，还是刚刚走上工作岗位的职场新人，此时学习小程序技术，正是最佳良机。你与有数十年编程经验的老手站在了同一起跑线上，因为小程序对所有人来说都是全新的技术。今天的菜鸟，未必就不能成为明日高手。

根据我在小程序培训中的观察，初学者最大的痛点是感觉技术太杂，要学的东西太多。买了一堆书堆在桌上，学完这个又学那个，难于将其融会贯通。行程未远，激情已耗大半。目前市面上还没有一本书从前端到后端、全面介绍小程序的开发技术，已有的书籍有的介绍了小程序组件而未介绍 JS 语言，有的介绍了 JS 语言却未讲解如何开发服务端程序，而本书首次全面介绍了小程序所需要用到的所有技术，从小程序组件到 WXSS 样式，从前端 JS 语言到后端 Go 语言，通过实战案例，由浅入深地介绍小程序开发涉及的所有内容，帮助读者快速成长为一名真正的微信全栈工程师。

读者对象

- 高校毕业生，中专技校毕业生。
- 工作 1～2 年的、渴望获得加薪技能的职场新人。
- 渴望以软件开发为谋生手段的自由“手艺”人。
- 准备报名或已参加小程序开发培训班的读者。

有人说，大学里最美好的事情就是找到一个喜欢的人，认认真真地谈一场无关名利的恋爱。但大学里不只有恋爱，在新学期伊始就开始学习小程序开发吧，这将是送给四年后的自己最好的礼物。许多人后悔在 2007 年第一款 iPhone 发布时没有开始学习 iOS 开发，只能羡慕那些早期的 iOS 开发者获得平台的初期红利。现在小程序来了，企业市场对小程

序的需求越来越旺，学好这门实用的技术，毕业后就不怕找不到工作；如果向往自由的生活，不想在公司打工，还可以自己接单，做 SOHO 一族。如果学得好，那么在校期间就可以接单，成为一名自食其力的编码“手艺人”。

如何阅读本书

本书主要包括四篇，内容分布如下。

- 第一篇，即第 1 章学前准备，讲解了小程序开发环境的准备及账号的注册。完成第 1 章的学习相当于取得了小程序技术殿堂的入场券。
- 第二篇，第 2～10 章，本篇是项目实战部分，其中第 2～6 章讲解小程序前端案例，使用了后台地址但未涉及后台编程；第 7～10 章在已有案例的基础上添加了后端程序的支持。先学习前端，再学习后端，每次专注一个点学习，更易理解和掌握。
- 第三篇，第 11～14 章，本篇详细地讲解了所有小程序组件的使用方法，所附示例几乎全部都是生产可用的，这就大大降低了初学者在美工上的学习门槛。
- 第四篇，第 15～17 章，本篇是综合练习部分，系统地介绍了 JS 语言、Go 语言、WXSS 样式语法等必备知识与技能。这 3 章既可作为工具手册，以备开发查询之需；每一章节又都有独立的练习代码，可便于读者利用课余或业余的碎片时间提高编码水平。

学习指引：

1. 读者从第 1 章开始到第 14 章，逐章学习，并运行测试所有的实例。每一章都附有源码，读者在学习的过程中如果遇到问题，可以下载作者的源码对照学习。

2. 待前 14 章全部学完，进入第 15～17 章的综合学习。在这个阶段的学习过程中，不妨直接用新学的知识直接深入修改前面业已完成的示例，将本书的示例变成自己的示例。如果有时间，建议将修改过程以博客的形式记录下来，并在社区发表，可以此加深印象。

小组学习

我至今最为怀念的时光，便是大学里和两位好友在机房里通宵学编程的日子。我们三个人相互鞭策又相互欣赏，经常比较谁的代码写得更优雅，谁的代码执行效率更高。

我希望每个读者都能找到朋友或同学组成一个学习小组，或 2 人，或 3 人，共同学习，相互激励，这样学习的效率和动力会高许多。孔子曰“三人行，必有我师”，诚不我欺。

勘误

由于作者水平有限，写作时间又很仓促，书中难免有不妥之处，恳请读者批评指正。

如果读者在阅读过程中发现了问题，或者有什么疑问，欢迎与作者联系。作者的邮箱是 liyi@rixingyike.com。

微信公众号

在学习本书的过程中，也欢迎加入作者的小程序微信群，关注微信公众号“艺述思维”，回复“小程序”就能加入。未来作者会举办读者线下交流会，请留意群内通知。

致谢

感谢机械工业出版社华章公司的杨绣国老师，她的认真和敬业令我折服。

感谢支持我进行《艺术论》创作的老杨同志、何超超、云哥、琥珀人生、杨龙晨等微信好友，感谢李萌、李书卫、南阳著名甲骨文书法家郝新安、国画家彭航、邯郸著名书法篆刻家杜文平、开封著名书法家王德云等 130 位日行一刻艺术天使们两年来的默默支持。

感谢所有读者，希望这本书对您的学习有所帮助。

石桥码农

2017 年 7 月于北京月亮河

目　录 *Contents*

第一篇 Part 1

基础入门篇

■ 第 1 章　学前准备

Chapter 1

第 1 章
学前准备

小程序是 2017 年 1 月 9 日微信推出的一种免安装、用完即走的轻 App。它基于微信环境运行，不需要用户安装。开发者可以基于微信开放的小程序技术规范，开发自己的小程序，并申请在微信上线，让自己的产品与微信 8 亿用户相连接。使用小程序技术开发的轻 App，具有入门学习简单、开发简便快捷、上线审核门槛低等优势。

工欲善其事，必先利其器。在开始学习小程序开发之前，需要先注册一个小程序账号，并在本机安装微信开发者工具。

1.1 注册账号

首先，在电脑上打开 https://mp.weixin.qq.com/，在页面右上角单击“立即注册”，如图 1-1 所示。

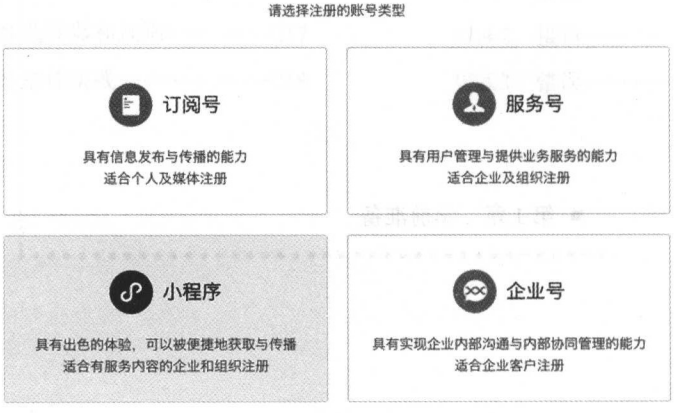


图 1-1

选择要注册的账号类型，即小程序。然后按要求填写账号信息，如图 1-2 所示。

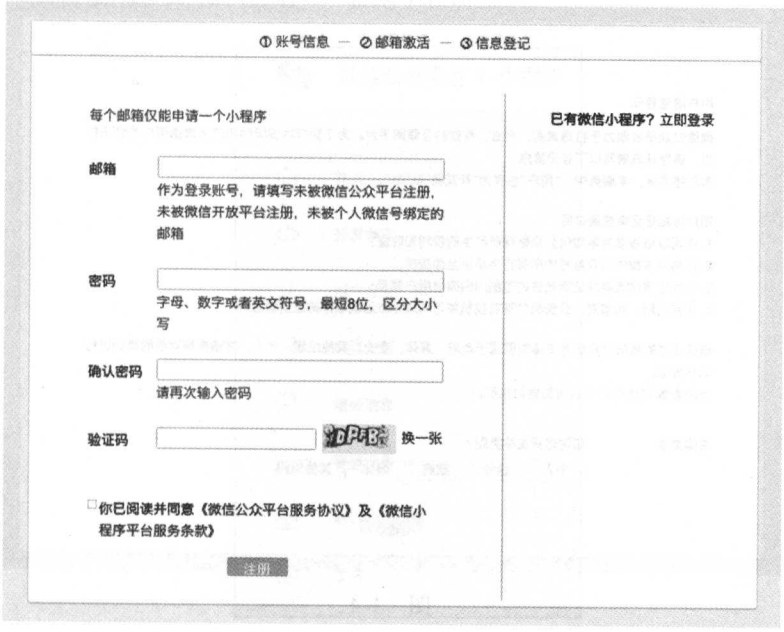


图 1-2

提交后，会看到激活账号的页面，如图 1-3 所示，上面显示已将确认邮件发送到之前注册的邮箱里。

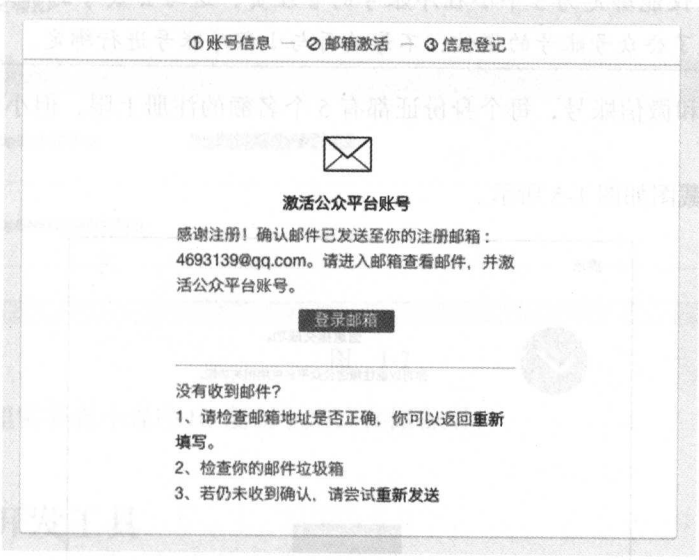


图 1-3

进入邮箱，打开收自“weixinteam”的邮件，单击激活链接，如图 1-4 所示。

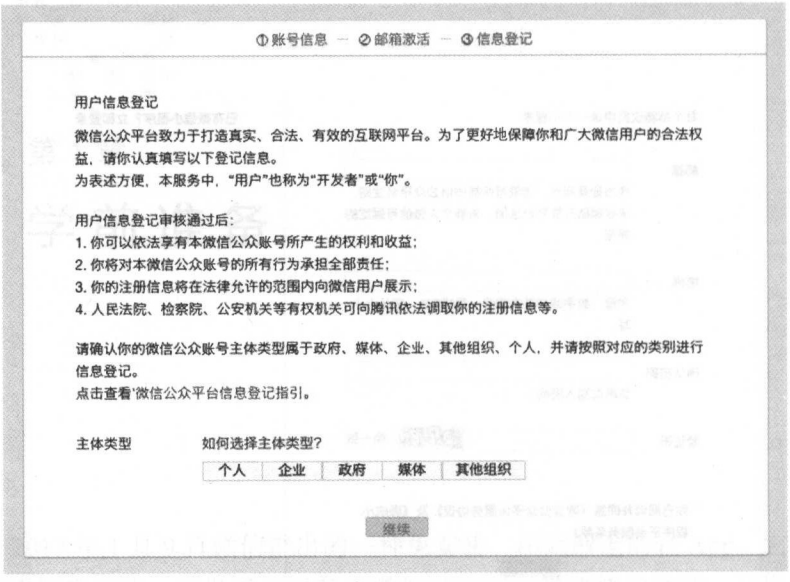


图 1-4

在“主体类型”的选项中选择“个人”，并填写相关信息。

在注册过程中，会用到一个微信账号来扫码验证身份。这个微信账号即为管理员账号，在以后的开发过程中会用到。

注意 每个微信仅能绑定为 5 个小程序账号的管理员，这与公众号的绑定限制是相同的。已经绑定了公众号账号的微信，不影响再与小程序账号进行绑定。

对于公众号和微信账号，每个身份证都有 5 个名额的注册上限，但小程序账号目前没有这个限制。

注册成功的截图如图 1-5 所示。

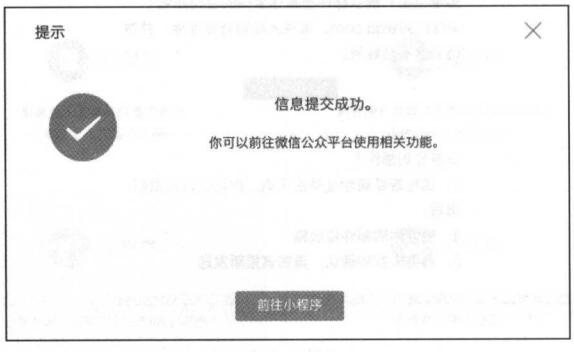


图 1-5

单击图 1-5 中的“前往小程序”按钮，自动登录小程序微信管理后台。
然后从左侧菜单中，选择“设置”，如图 1-6 所示。

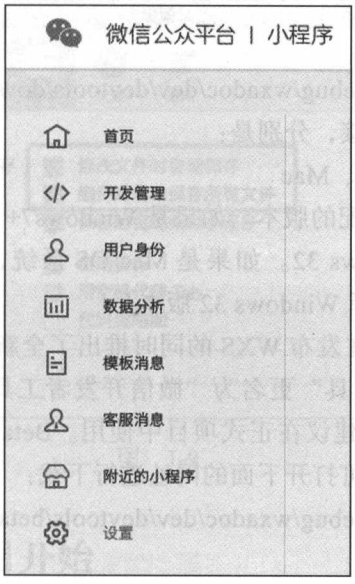


图 1-6

在设置面板中，选择“开发设置”标签，如图 1-7 所示。

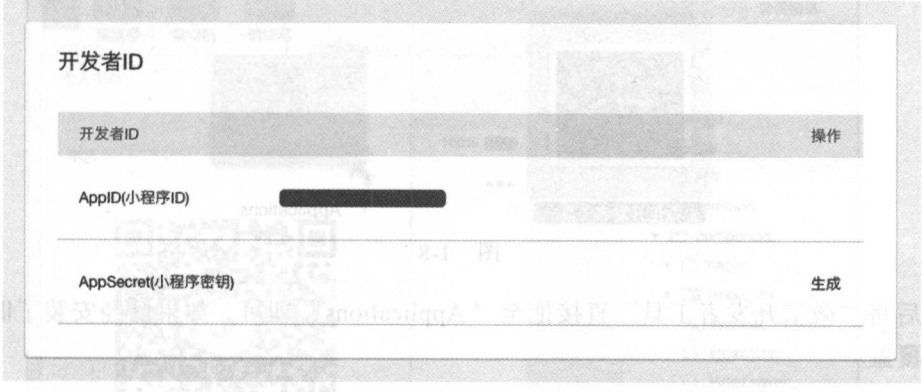


图 1-7

注意，这里需要将小程序 ID 复制下来，存储备用。

1.2 配置开发工具

微信开发者工具是微信官方推出的小程序开发工具，集代码编写、调试、效果预览等

功能于一体。

1.2.1 下载

打开下载网址[⊖]：

https://mp.weixin.qq.com/debug/wxadoc/dev/devtools/download.html

会看到三个版本的下载链接，分别是：

Windows 64、Windows 32、Mac

选择与自己的电脑系统适配的版本。如果是 Windows7+ 系统，则选择 Windows 64。如果是 XP 系统，则选择 Windows 32。如果是 Mac OS 系统，则选择 Mac。如果使用的是 Windows 7# 32 位系统，则选择 Windows 32 版本。

2017 年 8 月 22 日，微信在发布 WXS 的同时推出了全新界面的微信开发者工具的 Beta 版本，将“微信 Web 开发者工具”更名为“微信开发者工具”。Beta 版本用于优先发布新特征、新组件、新功能，但不建议在正式项目中使用。Beta 版本可与正式版本同时安装在一台机器之上，感兴趣的读者可打开下面的网址进行下载：

https://mp.weixin.qq.com/debug/wxadoc/dev/devtools/beta.html

1.2.2 安装

此处小程序的安装，将以 Mac OS 系统为例进行讲解。获得 dmg 安装包之后，双击打开，如图 1-8 所示。

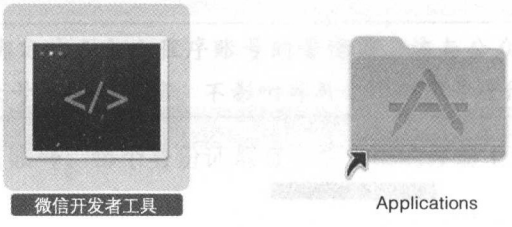


图 1-8

然后将“微信开发者工具”直接拖至“ Applications”即可。如果已经安装了旧版本，则选择覆盖。

1.2.3 设置编辑器属性

安装完成后，就可以设置编辑器的属性了。以 Mac 为例，依次打开“菜单”→“设置”→“编辑设置”，如图 1-9 所示。

在图 1-9 中，要同时选中“修改文件时自动保存”和“编译时自动保存所有文件”。单

⊖ 手动输入地址比较麻烦，可以在微信公号“艺述思维”中发送“小程序开发工具下载”得到相关链接。

击“保存”按钮退出，这样设置可以减少开发过程中手动保存代码的麻烦，此处的设置对所有项目都生效。

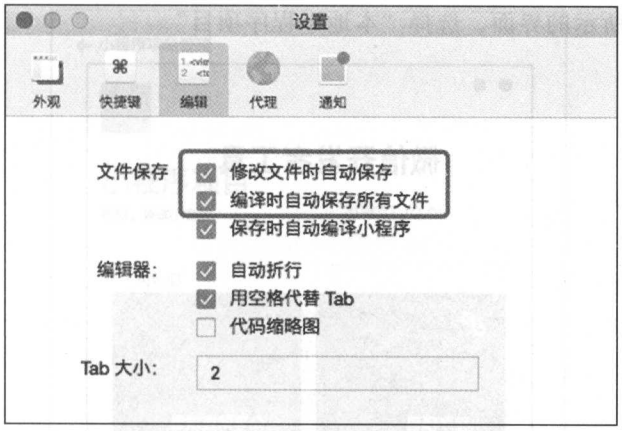


图 1-9

1.3 从 quick start 项目开始

现在，可以启动微信开发者工具了，选择“程序项目”，启动后的界面如图 1-10 所示。

图 1-10

如果未曾登录，使用管理员微信账号，扫码就可以登录“微信开发者工具”了。管理员微信账号是在 1.1 节注册小程序账号时所用的微信账号。

1.3.1 创建项目

若要创建新的项目，可通过如下步骤来实现。

首先在图 1-11 所示的界面，选择“本地小程序项目”。

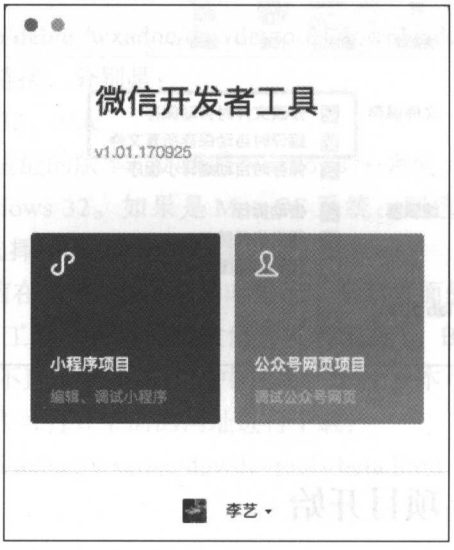


图 1-11

然后单击下方的“+”按钮增加新的项目，如图 1-12 所示。

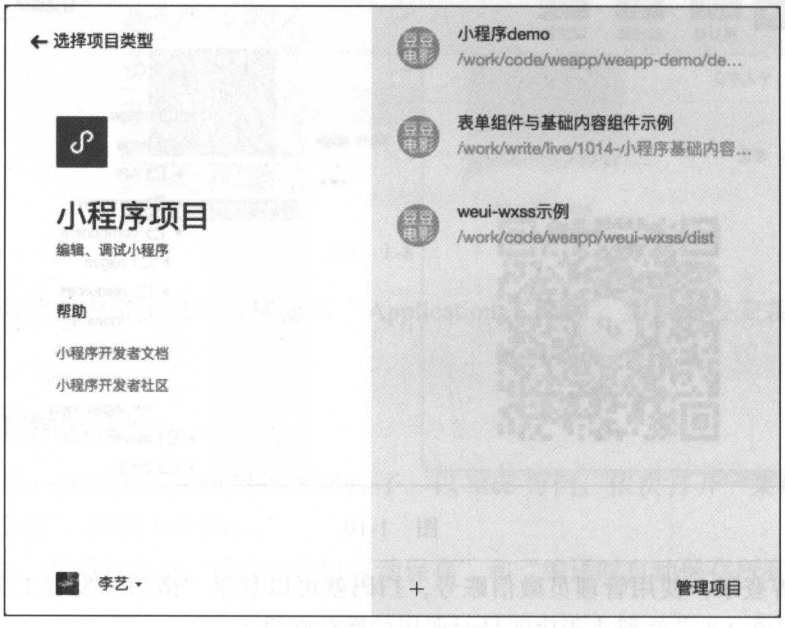


图 1-12

之后会进入图 1-13 所示的界面，在其中的 AppID 中填写小程序 ID，即在 1.1 节从小程序微信后台复制的字符串。至于项目名称，可随意填写，例如“小白从 0 到 1 学开发”。

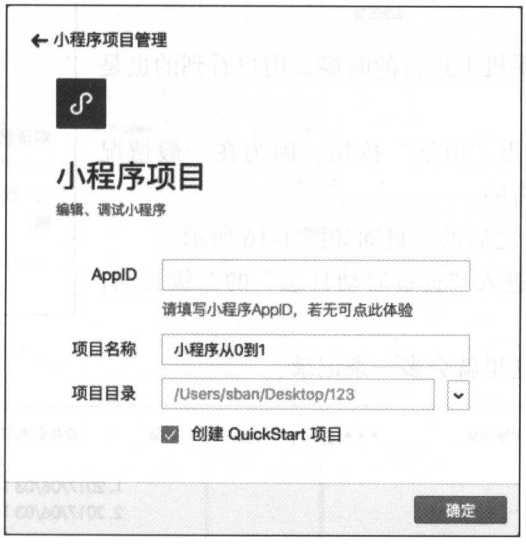


图 1-13

之后在这里选择一个空目录，如图 1-14 所示。

图 1-14

因为只有选择了空目录，才能出现“在当前目录中创建 quick start 项目”这个选项，默认是勾选的，默许这个设置。

单击“添加项目”，即可完成 quick start 项目的初建。

1.3.2 运行项目

首次运行 quick start 项目时，程序会提示授权，如图 1-15 所示。

图 1-15

允许这个请求。在手机上运行的时候，用户看到的也是类似的提示。

这个地方很容易错点“拒绝”按钮，因为在一般情况下，主操作按钮总是居右的。

quick start 项目运行之后的主页面如图 1-16 所示。

单击自己的头像，进入“查看启动日志”的二级页面，如图 1-17 所示。

每启动一次项目，这里就会多一条记录。

图 1-16

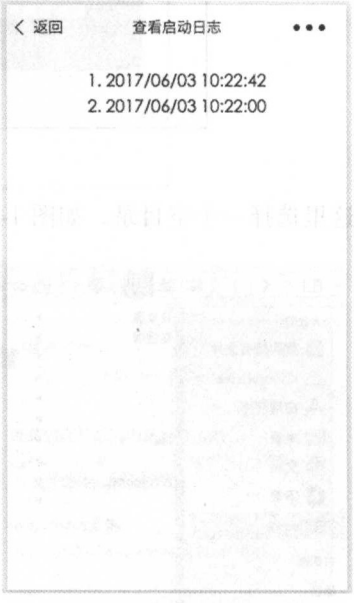


图 1-17

1.3.3 刷新项目

要刷新项目，可单击“微信开发者工具”左侧工具栏中的刷新按钮，如图 1-18 所示。

或者按 <Command+B> 组合键（Windows 用户按 <Ctrl+B>），重启项目。再次查看上面的日志页面，便多了一条记录。

本章完成了小程序账号的注册以及开发工具的安装和配置，创建了“quick start”项目。“quick start”项目相当于其他编程语言中的“Hello world”程序，旨在帮助初学者快速建立对小程序开发的认知。开发者也可以基于“quick start”项目开发自己的项目。从第 2 章

开始，本书将进入小程序项目实战。

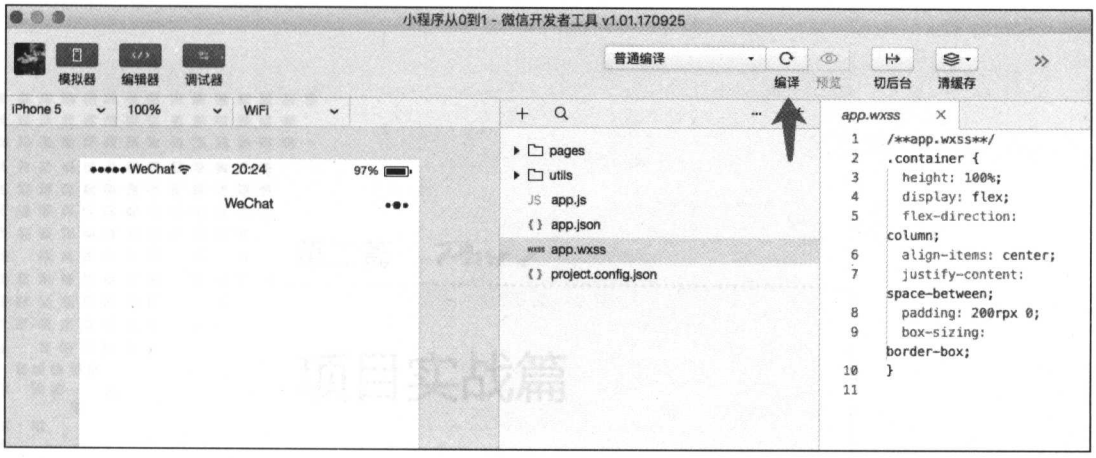
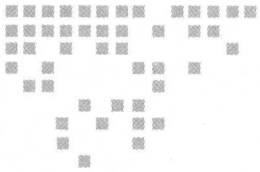


图 1-18

第二篇 Part 2

项目实战篇



Chapter 2 第 2 章

豆豆电影

本章将调用豆瓣接口，实现电影榜单的展示，以及检索、实时检索、信息展示等功能。这一章仅讲解前端操作，不涉及后端 Go 语言编程。在学习过程中，如果对个别的关于 JS、WXSS 的概念理解比较困难，可跳至第 15 章和第 16 章查看相关内容。

2.1 从 splash 功能开始

首次进入某些 App 时，通常在界面的底部有三个面板指示点，分别对应于三张图片，可以左右滑动查看。

本节将以实现这个功能来开启小程序的实践之旅。

2.1.1 创建项目

接下来开始逐步实现 splash 功能，首先打开微信开发者工具，单击“+”号按钮创建新项目，如图 2-1 所示。

相关参数在图 2-1 中已有展示，需要注意的是，“项目目录”必须选择一个空目录，“AppID”和“项目名称”需要根据自己的实际情况来填写，然后单击“添加项目”按钮。

之后，单击工具栏中的“详情”按钮，得到图 2-2 所示的界面。

在图 2-2 中，选中框线中的所有复选框，特别是其中的“开发环境不检验请求域名”、TLS 版本以及 HTTP 证书。如果不选中该复选框，那么请求豆瓣 API 将会失败。

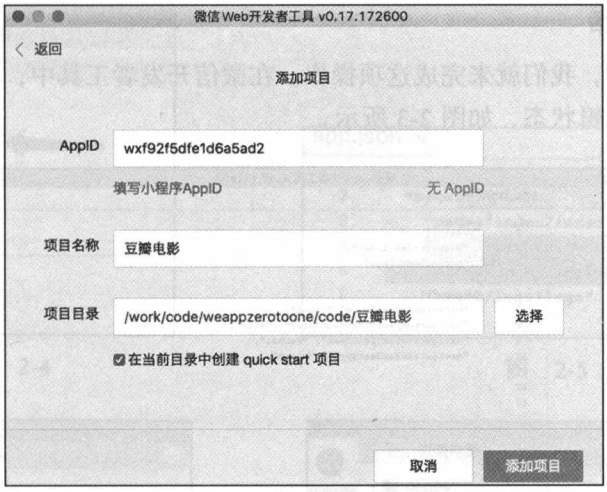


图 2-1

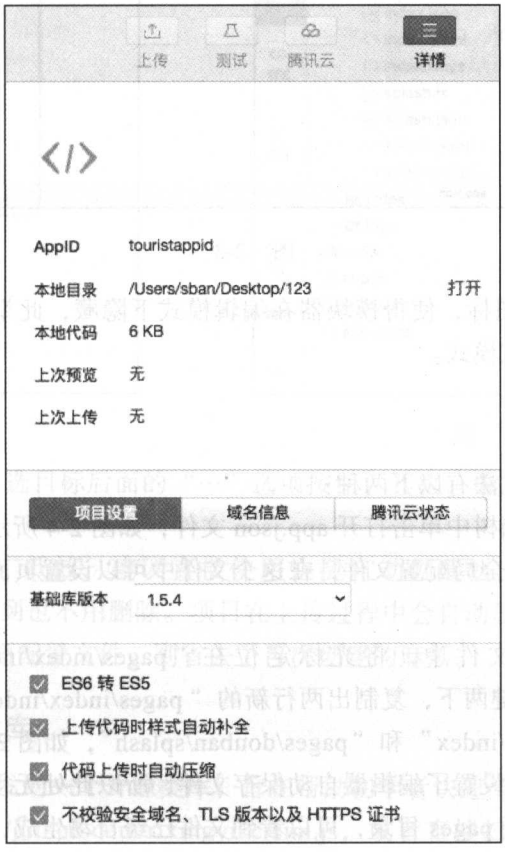


图 2-2

操作完成，至此即成功创建了一个新的项目。

2.1.2 隐藏模拟器

在创建项目之后，我们就来完成这项操作。在微信开发者工具中，单击工具栏中的“编辑”按钮，切换到编辑状态，如图 2-3 所示。

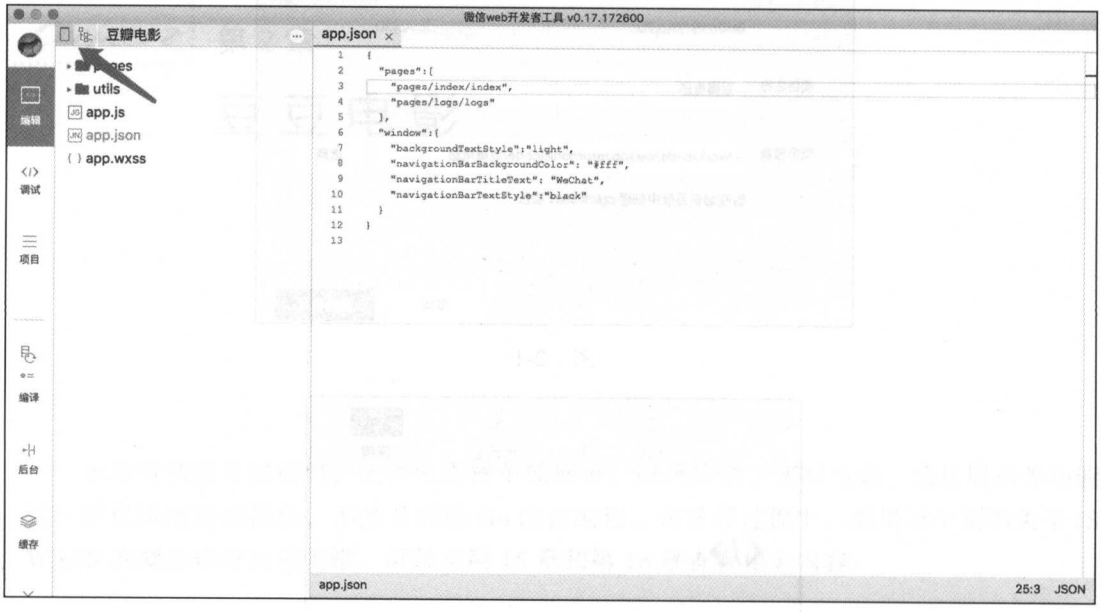


图 2-3

单击左上角的手机图标，使得模块器在编辑模式下隐藏，此举将便于编辑代码。如果需要预览，请切换至调试模式。

2.1.3 快捷创建页面

创建新页面的常用方法有以下两种：

第一种方法是在文档树中单击打开 app.json 文件，如图 2-4 所示。

app.json 是小程序的全局配置文件，在这个文件中可以设置页面路径、窗口样式、网络超时时间及 tarBar 等。

在打开的 app.json 文件中，将光标定位在“pages/index/index”这一行，同时按<Alt+Shift+Down>组合键两下，复制出两行新的“pages/index/index”。将新复制的两行分别修改为“pages/douban/index”和“pages/douban/splash”，如图 2-5 所示。

因为在 1.2 节中已经设置了编辑器自动保存文件，所以此处无须保存。

在文档树中单击展开 pages 目录，可以看到文件已经自动生成，如图 2-6 所示。

使用这种方法可以显著提高新建 page 页面文件的效率。

第二种标准的新建页面的方法如图 2-7 所示。

图 2-4

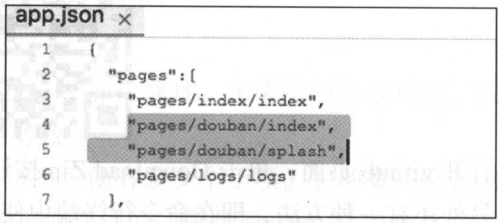


图 2-5

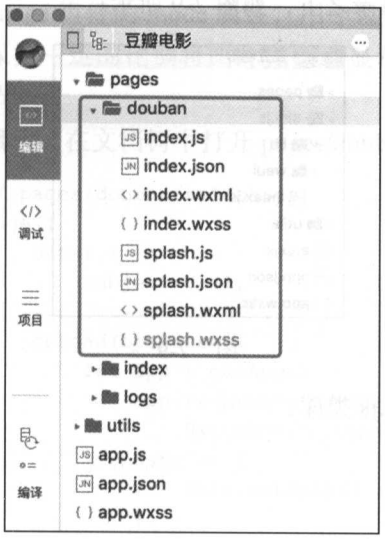


图 2-6

图 2-7

首先，在文档树中点选目标后面的“…”选项按钮；其次，单击“新建”；最后，选择新建的文件类型。

这种方法比较麻烦、低效，建议直接使用快捷自动创建法。至于自动生成的 .json 与 .wxss 文件，如果用不到也不用删除。项目在上传过程中会自动压缩，无须介意这些空文件。如果要用到 .json 页面配置文件，则省去了再次创建的麻烦。

2.1.4 引用 sim.js 类库

sim.js 类库是笔者开发的一个开源类库，旨在帮助初学者快速开发小程序前端。

在电脑上打开 https://github.com/rixingyike/sim.js，或者直接扫描下方的二维码，下载 sim.js 类库压缩包。

打开 github 页面，单击 Download Zip 按钮，下载源码包，如图 2-8 所示。

另外还有一种方法，即在命令行终端中使用 git clone 指令下载源码，这种方法更普遍，稍后会有介绍。

现在将下载的源码，解压至豆豆电影小程序的根目录之下，如图 2-9 所示。

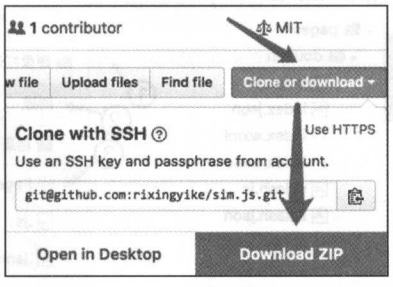


图 2-8

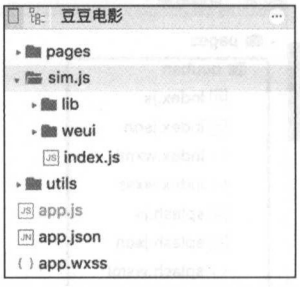


图 2-9

然后在文档树中打开 app.js 文件，在顶部引用 sim.js 类库：

```
let app = require("./sim.js/index.js")
```

将第 4 行代码

```
App({
```

替换为

```
App(Object.assign(app,{
```

将最后一行替换为：

```
}))
```

经过以上步骤，sim.js 引入完毕。这两步注入将 sim.js 类库提供的工具类方法，附在了 App 对象之上。在接下来的开发中，通过 getApp() 来获取 App 对象将能通过它使用 sim.js 类库提供的这些方法。

let 是 JS 语言声明变量的关键字，关于变量的更多信息，请参见本书的 15.1.1 节。

2.1.5 实现 splash 效果

首先，在文档树中打开 pages/douban/splash.wxml 文件，将代码修改为：

```
<swiper style="height: 100%;width: 100%;" indicator-dots>
    <swiper-item wx:for="{{ subjects }}" wx:key="{{ item.id }}" style="flex: 1;">
        <image src="{{ item.images.large }}" mode="aspectFill" style="position:
            absolute;height: 100%;width: 100%;opacity: .8;" />
    </swiper-item>
</swiper>
```

这里使用了 swiper 组件，其中的 indicator-dots 是布尔属性，不需要填写任何值，仅仅放在那里就能表示其值为 true，与填写任何值均等效。

swiper-item 是 swiper 组件的子项，有多少张图片便有多少个子项。它的宽高会被自动设置为 100%，所以无须重复设置。对全屏起决定性作用的，则是 swiper 的 style 属性，它将宽高均设置为 100%。关于 swiper 组件的更多内容，请参见本书的 11.3 节。

image 组件的 mode 属性，用于设置图像的缩放策略。aspectFill 是最常用的缩放模式，它会保持一定的比例将图片完整地显示出来。关于 image 组件的更多内容，请参见本书的 14.3 节。

然后，在文档树中打开 pages/douban/splash.js 文件，将代码修改为：

```
// pages/douban/splash.js
Page({
    data: {
        subjects: [],
    },
    onLoad(options) {
        let app = getApp()
            app.request("https://api.douban.com/v2/movie/coming_soon?start=
                0&count=3").then(
            data => {
                this.setData({ subjects: data.subjects })
            }
        )
    }
})
```

在上述代码中，subjects 是 wxml 代码中用于数据绑定的数组。这里使用了 app.request 方法，用于请求豆瓣的 API。获取到数据之后，即可使用 setData 方法渲染视图。这里的 setData 会实现如下两个功能。

- 保存数据至 subjects 变量。
- 通知页面视图，若数据有更新，则进行渲染。

在 app.request() 方法返回的对象上调用 then 方法，是处理接口调用成功之后的逻辑。then 方法接受一个函数为参数，笔者在这里传入的是使用箭头符号简写的匿名函数。关于箭头函数，参见本书的 15.2.9 节。

在 2.1.4 节引用 sim.js 类库，就是为了此时在这里使用。2.1.4 节通过注入方法修改了默认的 App 定义，是为了把 sim.js 类库的 request 方法加到 App 对象上。选择 App 对象注入，

可以最大限度地减少项目对框架的注入依赖。

在任何页面，只要能通过 getApp() 取得全局唯一的 App 对象，就可以使用 sim.js 类库的功能。

1. 如何设置首页

在文档树中打开 app.json 文件，将光标定位到图 2-10 所示的这一行。

同时按下 <Command+Up>（或 <Ctrl+Up>）组合键，将“pages/douban/splash”移至首行，因为首行意味着首页。

app.json ×

```
1  {
2    "pages":[
3      "pages/index/index",
4      "pages/douban/splash",
5      "pages/douban/index",
6      "pages/logs/logs"
7    ],
```

图 2-10

2. 如何添加样式

此时已将“pages/douban/splash”设置为首页，同时按下 <Command+B> 组合键（或 <Ctrl+B> 组合键）刷新项目，什么都看不到。为什么？接下来将通过调试面板来寻找原因。

在调试工具区域，切换至 wxml 面板，可以看到已经渲染了三个 swiper-item，如图 2-11 所示。

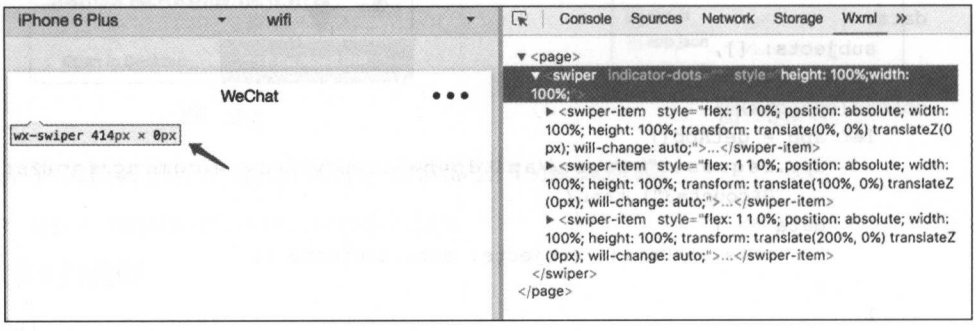


图 2-11

但是 swiper 的高度为 0。这是因为在小程序中，page 容器的高度默认是 0。将 swiper 组件的高度样式设置为 100% 是无用的，因为父容器没有高度。

解决办法很简单，在文档树中打开 app.wxss 文件，在尾部添加以下样式：

```
page{
    height: 100%;
    background-color: #f9f9f9;
}
```

这里除了高度以外，还添加了一个浅灰色的背景色。浅灰色的默认背景有助于设计产品的 UI。

再次按下 <Command+B> 组合键（或 <Ctrl+B> 组合键）刷新项目，功能已然正常。之前没有显示，是因为 swiper 容器没有获得足够的高度，我们通过调试面板查出了问题所在。

善用调试面板，有助于在开发中快速找到出错的缘由。

2.1.6 下载源码

在本节的学习过程中，如果遇到问题，可以通过如下方式来解决。

- 加入小程序微信群与作者以及其他读者一同探讨。微信扫描前言中的二维码，关注“艺述思维”，发送“小程序”进群。
- 下载作者的源码，对照查找问题。在公众号发送“豆豆电影 2.1”获取下载地址。

2.2 缓存本地数据

通常，首次进入小程序应用时，会展示全屏图片滑动，第二次进入时则不再展示，本节就来实现这个功能。

2.2.1 使用 wx.setStorage 接口

接口 wx.setStorage 用于从本地缓存中异步获取指定 key 所对应的内容。其与接口 wx.getStorage 相对应。

在文档树中打开 app.json 文件，将 pages/douban/index 调整为首页。

打开 pages/index/index.js 源码，修改 onLoad 函数，代码如下：

```
onLoad(options) {
    wx.getStorage({
        key: 'has_shown_splash',
        fail: err => {
            wx.redirectTo({
                url: '/pages/douban/splash',
            })
        }
    })
}
```

其中，wx.redirectTo 是页面跳转接口，用于从当前页面跳转至 pages/douban/splash 页面。

打开 pages/douban/splash.js 源码，在 onLoad 末尾添加如下代码：

```
wx.setStorage({
    key: "has_shown_splash",
    data: true
})
```

这里的 has_shown_splash 是键名，必须与 pages/index/index.js 中的代码保持一致。

2.2.2 使用 Storage 面板

按 <Command+B> 组合键（或 <Ctrl+B> 组合键）刷新项目，程序首先会进入 pages/douban/

index，随后会跳转到 pages/douban/splash 页面。再次刷新，本地已有记忆，因此不会再看到 splash 页面。那么该功能是如何实现的呢？下面一起来看一下。

在调试工具区域，打开 Storage 面板，如图 2-12 所示。

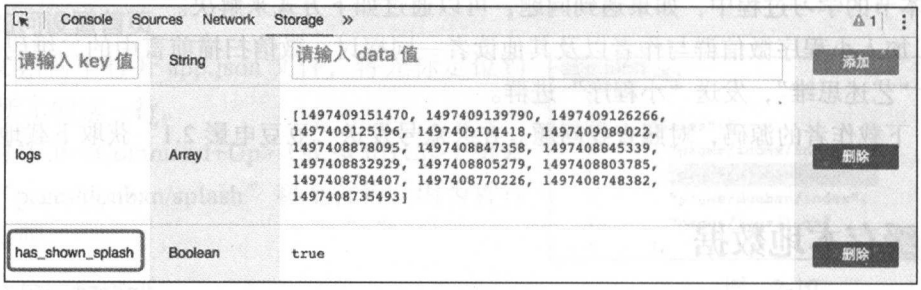


图 2-12

从这个面板中可以看到本地缓存中保存的所有数据，包括保存在 pages/douban/splash 页面的 has_shown_splash 变量，这就是本地已有的记忆数据。

若单击“删除”按钮，再按 <Command+B> 组合键（或 <Ctrl+B> 组合键）刷新项目，就又可以看到 splash 全屏图片滑动的效果了。

has_shown_splash 变量上面的一行是 quick start 项目默认生成的代码所记录的缓存，相关代码在 pages/logs/logs.js 中。

2.2.3 省略 function 关键字

在快捷方法自动生成的页面 js 代码中，onLoad 函数默认是这样的：

```
onLoad: function (options) {
}
```

而笔者声明的 onLoad 是这样的：

```
onLoad(options) {
}
```

两者的不同之处在于，笔者的声明没有使用 function 关键字。

笔者使用的是 ES6 语法，使用 ES6 语法不仅可以减少键盘作业，还可以在匿名函数内部使用 this 关键字。如下所示，方法 then 接受的参数即为匿名函数：

```
app.request("https://api.douban.com/v2/movie/coming_soon?start=0&count=3").then(
    data => {
        this.setData({ subjects: data.subjects })
    }
)
```

其中，data => {} 是一个 ES6 语法声明的匿名函数。data 是匿名函数的形参。旧式写法是这样的：

```
onLoad(options) {
    let app = getApp()
    let that = this

    app.request("https://api.douban.com/v2/movie/coming_soon?start=0&count=3").
        then(
        function(data) {
            that.setData({ subjects: data.subjects })
        }
    )
}
```

如上述代码所示，如果不使用ES6语法，则需要先于匿名函数之外声明一个that变量指向this，才能在匿名函数内部访问真正的this对象。

在匿名函数内部直接使用this对象，将无法调用其setData方法。匿名函数内部的this对象，并不是当前页面的Page对象。读者可以自行测试一下。

注意 在2.1.1节中，我们选择了项目属性面板中的“开启ES6转ES5”选项，只有这样才能在项目中使用ES6语法。每次新建一个项目时，都不妨首先开启这个选项。

2.3 实现页首splash效果

本节将学习如何在小程序页首实现图2-13所示的效果。

图 2-13

2.3.1 使用swiper组件

2.1.5节使用swiper组件实现了全屏的splash效果，本节将再次使用这个组件实现页面顶部的splash效果。

切换至编辑模式，在文档树中打开pages/douban/index.wxml文件，修改代码为：

```
<swiper style="height:450rpx" indicator-dots autoplay="true" interval="5000"
    duration="1000">
    <swiper-item wx:for="{{ boards[0].movies }}" wx:key="{{ item.id }}">
        <navigator hover-class="none">
            <image style="height:450rpx;width:750rpx;" src="{{ item.images.large }}"
                mode="aspectFill" />
        </navigator>
    </swiper-item>
</swiper>
```

这里使用的仍然是swiper组件，与2.1.5节的不同之处在于，此次设置height为

450rpx。rpx 是 responsive pixel 的简称，是小程序开发中实现屏幕自适应 UI 的长度单位。在页面设计上，微信小程序规定屏幕的宽恒为 750 rpx。iPhone6 屏幕的宽度为 375px，换算过来则是 1rpx=0.5px，以此设计 UI 元素的尺寸。其他型号的手机，1rpx 所代表的宽度将视屏宽而定。

此处设置 swiper 高度为 450rpx，比屏宽的一半略高，这也是常见的设计比例。

在文档树中打开 pages/douban/index.js 页面，修改 data 声明为：

```
data: {
    boards: [{ key: 'in_theaters' }, { key: 'coming_soon' }, { key: 'top250' }],
},
```

其中，“in_theaters”“coming_soon”等是豆瓣 API 需要用到的参数。

2.3.2 批量调用接口

在当前页面“pages/douban/index”的逻辑层代码中，我们需要调用豆瓣接口三次拉取三个不同的榜单数据。在全部拉取完成之后，再调用 setData 方法渲染页面。这种场景比较适合使用 sim.js 类库提供的 app.promise.all 方法批量调用接口。

批量调用接口，可采用如下方式。

添加一个 retrieveData 函数：

```
retrieveData() {
    let app = getApp()

    var promises = this.data.boards.map(function (board) {
        return app.request(`https://api.douban.com/v2/movie/${board.key}?start=
            0&count=10`)
            .then(function (d) {
            if (!d) return board
            board.title = d.title
            board.movies = d.subjects
            return board
        }).catch(err => console.log(err))
    })

    return app.promise.all(promises).then(boards => {
        if (!boards || !boards.length) return
        this.setData({ boards: boards, loading: false})
    })
}
```

这个函数主要完成如下两件事情。

- 依据参数不同，从豆瓣 API 拉取三次列表。
- 待三次拉取全部完成之后，调用 setData 设置数据通知页面渲染。

sim.js 类库中复合了 bluebird 类库，这是一个 Promise 类库，主要提供了如下四种功能。

- 使用 then 实现链式调用。
- 使用 Promise.all 实现并行调用。
- 使用 Promise.race 实现竞赛调用。
- 使用 catch 捕捉异常。

sim.js 框架将 Promise 对象注入到了 App 对象上，所以在这里可以使用 app.promise.all 进行并行调用。它与直接使用 Promise.all 的效果是等同的，但免去了在 Page 页面中再次引用 js 类库的麻烦。关于 Promise，稍后会有更多的介绍。

代码中出现的 then 的用法，在 2.1.5 节已经讲过，它表示异步调用成功之后应执行什么。

2.3.3 使用 wx.getStorage 接口

在 2.1.5 节，笔者在“pages/douban/splash”页面向本地缓存中存入了“has_shown_splash”变量，用于标识已经进入过 splash 页面。本节将尝试取出这个变量，如果不为空，则调用“retrieveData”函数；如果为空，则跳转至“pages/douban/splash”页面。

在 onLoad 函数中增加对 retrieveData 的调用，代码如下所示：

```
onLoad(options) {
    wx.getStorage({
        key: 'has_shown_splash',
        success: res => {
            this.retrieveData()
        },
        fail: err => {
            wx.redirectTo({
                url: '/pages/douban/splash',
            })
        }
    })
}
```

success 是接口 wx.getStorage 异步调用成功后调用的函数。如果是首次进入小程序，没有找到缓存，则进入 pages/douban/splash 页面；反之，则调用自建的 retrieveData 函数，批量拉取豆瓣数据。

2.3.4 下载源码

在本节的学习过程中，如果遇到问题，可以通过如下方式来解决。

- 加入小程序微信群与作者及其他读者一同探讨。微信扫描前言中的二维码，关注“艺述思维”，发送“小程序”进群。
- 下载作者的源码，对照查找问题。在公众号中发送“豆豆电影 2.3”获取下载地址。

2.4 实现横向滑动列表

本节将实现可以左右滑动的列表，如图 2-14 所示。

图 2-14

2.4.1 列表渲染

在文档树中打开 pages/douban/index.wxml 文件，添加如下代码：

```
<view wx:for="{{ boards }}" wx:key="{{
    item.key }}" class="weui-panel weui-
    panel_access">
    <view class="weui-panel__hd">
    {{ item.title }}
    </view>

    <view class="weui-panel__bd">
        <view style="padding:10px" class="weui-media-box weui-media-box_appmsg"
            hover-class="weui-cell_active">
            <scroll-view scroll-x>
                <view style="display:flex;">
                    <navigator wx:for="{{ item.movies }}" wx:key="{{ item.id }}">
                        <view style="display:flex;flex-direction:column;width:180r
                            px;margin:10rpx;">
                            <image style="width:180rpx;height:250rpx;" src="{{
                                item.images.large }}" mode="aspectFill" />
                            <text style="text-align:center;overflow:hidden;white-
                                space:nowrap;text-overflow:ellipsis;font-size:13px;
                                padding-top:5rpx;">{{ item.title }}</text>
                        </view>
                    </navigator>
                </view>
            </scroll-view>
        </view>
    </view>

    <view class="weui-panel__ft">
        <navigator class="weui-cell weui-cell_access weui-cell_link">
            <view class="weui-cell__bd">更多</view>
            <view class="weui-cell__ft weui-cell__ft_in-access"></view>
        </navigator>
    </view>
</view>
```

其中，scroll-view 组件用于展示一个可滚动区域，scroll-x 属性代表横向滚动。

若要实现横向滚动，则容器内容必须超过屏幕宽度。在子视图 view 中，设置 display 样式为 flex，flex 是 Flexible Box 的缩写，意为“弹性布局”，设置为此属性，即 view 内容

可以向右无限伸展。

wx:for 用于将数组渲染在视图上，数组有几项便渲染几行。从 2.3.2 节的代码可以得知，boards 数组有三个元素，所以这里会渲染三个附有“weui-panel”样式的 view。

wx:for 可以嵌套使用，在上面的 wxml 代码中，横向列表的渲染是通过嵌套渲染来实现的。被循环数组的当前项的默认变量名称是 item，在被嵌套的 wx:for 里，子 item 会将父 item 覆盖。

在图 2-15 中，第一个 item 指的是数组 boards 的子项，第二个及其后面的 item 指的是 boards.movies 的子项。

```
18        <view style="display:flex;">
19          <navigator wx:for="{{ item.movies }}" wx:key="{{ item.id }}">
20            <view style="display:flex;flex-direction:column;width:180rpx;margin:10rpx;">
21              <image style="width:180rpx;height:250rpx;" src="{{ item.images.large }}" mode="aspectFill" />
22              <text style="text-align:center;overflow:hidden;white-space:nowrap;text-overflow:ellipsis;
    font-size:13px;padding-top:5rpx;">{{ item.title }}</text>
23            </view>
24          </navigator>
25        </view>
```

图 2-15

2.4.2 引用样式

在文档树中打开 app.wxss 文件，在顶端添加如下代码：

```
@import 'sim.js/weui/weui.wxss';
```

这里的 @import 是 CSS 引用另一个样式文件的语法，它与 wxml 中使用的 import 并不相同。

weui 是一个样式库，非常符合微信小程序设计指南的要求，应用在项目中可以显著提高初学者的开发速度。

2.4.3 下载源码

在本节的学习过程中，如果遇到问题，可以通过如下方式来解决。

- 加入小程序微信群与作者及其他读者一同探讨。微信扫描前言中的二维码，关注“艺述思维”，发送“小程序”进群。
- 下载作者的源码，对照查找问题。在公众号中发送“豆豆电影 2.4”获取下载地址。

2.5 实现电影详情页

单击主页列表中的单项，会进入电影详情页，并展示电影的一些基本信息，如图 2-16 所示，本节就来实现这样的功能。

2.5.1 格式化代码

使用 2.1.3 节快捷创建页面的方法，添加 pages/douban/item 页面，用于展示电影详情。

在文档树中，打开 pages/douban/item.wxml 文件，添加如下代码，引入 weui 样式库：

```
@import 'sim.js/weui/weui.wxss';
```

如果读者是从笔者的源码中复制的代码，那么在复制粘贴之后，应在文档区右击从弹出的右键菜单中选择“格式化代码”命令，这样可以快速将代码格式化，易读易改，如图 2-17 所示。

图 2-16

2.5.2 逻辑层

在文档树中打开 pages/douban/item.js 文件，修改 data 变量为：

```
data: {
    loading: true,
    movie: {}
}
```

将 onLoad 函数修改为：

```
onLoad(options) {
    let app = getApp()

    app.request(`https://api.douban.com/v2/movie/subject/
        ${options.id}`)
        .then(d => {
        this.setData({ movie: d, loading: false });
        wx.setNavigationBarTitle({ title: d.title });
    }).catch(e => {
        console.error(e);
    })
}
```

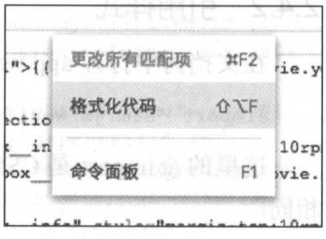


图 2-17

在上述代码中，使用 Promise 方式从豆瓣 API 拉取数据，然后在 then 回调中取得数据并绑定。

loading 变量用于指示数据是否加载完成，可使用它在页面上显示一个加载提示。

options 是从其他页面传递过来的参数集合，可从中获取 id，这将是 2.5.4 节 wxml 标签中自定义的参数名称。这里使用电影的 id，从豆瓣 API 中拉取单个电影的详细信息。

2.5.3 视图层

在文档树中打开 pages/douban/item.wxml 文件，修改标签代码为如下形式：

```
<loading hidden="{{ !loading }}">加载中</loading>
<image style="position: fixed;left: 0;top: 0;right: 0;bottom: 0;height: 100%;width:
    100%;z-index: -1000;opacity: .05;" src="{{ movie.images.large }}" mode=
    "aspectFill" />
<scroll-view scroll-y>
    <view class="weui-article">
        <view class="weui-article__section">
            <image class="weui-article__img" src="{{ movie.images.large }}"
                mode="aspectFit" style="width: 100%;height: 800rpx" />
        </view>
        <view class="weui-article__h1">{{ movie.title }}({{ movie.year }})</view>

        <view class="weui-article__section">
            <view class="weui-media-box__info" style="margin-top:10rpx;">
                <view class="weui-media-box__info__meta">评分: {{ movie.rating.
                    average }}</view>
            </view>
            <view class="weui-media-box__info" style="margin-top:10rpx;">
                <view class="weui-media-box__info__meta">导演:
                    <block wx:for="{{ movie.directors }}" wx:key="{{ item.id }}">
                        {{ item.name }} </block>
                </view>
            </view>
            <view class="weui-media-box__info" style="margin-top:10rpx;">
                <view class="weui-media-box__info__meta">主演:
                    <block wx:for="{{ movie.casts }}" wx:key="{{ item.id }}"> {{
                        item.name }} </block>
                </view>
            </view>
        </view>

        <view class="weui-article__section">
            <view class="weui-article__p">
                {{ movie.summary }}
            </view>
        </view>
    </view>
</scroll-view>
```

loading 组件并没有出现在微信官方文档中，但它仍然可以使用绑定变量的方式来控制它的显示。如果组件的属性是一个布尔值，那么只有使用这样的绑定方法才可能使它的值为 false：

```
hidden="{{ !loading }}"
```

这里再一次使用了 scroll-view 组件，但是 scroll-y 为 true，代表允许纵向滚动。

2.5.4 页面跳转

在文档树中打开 pages/douban/index.wxml 文件，修改 wxml 标签，在 scroll-view 容器内，在 navigator 组件上添加 url 属性，如图 2-18 所示。

```
15    <view class="weui-panel__bd">
16      <view style="padding:10px" class="weui-media-box weui-media-box_appmsg" hover-class="weui-cell_active">
17        <scroll-view scroll-x>
18          <view style="display:flex;">
19            <navigator url="item?id={{item.id}}" wx:for="{{ item.movies }}" wx:key="{{ item.id }}">
20              <view style="display:flex;flex-direction:column;width:180rpx;margin:10rpx;">
21                <image style="width:180rpx;height:250rpx;" src="{{ item.images.large }}" mode="aspectFill" />
22                <text style="text-align:center;overflow:hidden;white-space:nowrap;text-overflow:ellipsis;
    font-size:13px;padding-top:5rpx;">{{ item.title }}</text>
23              </view>
24            </navigator>
25          </view>
26        </scroll-view>
27      </view>
28    </view>
```

图 2-18

其中的 id 是在 2.5.2 节 js 代码中获取到的参数。item 是页面名称，因为此页与 item 页面位于同一个目录之下，所以可以使用相对地址。

在页面顶部的 swiper 组件中，在 navigator 组件上添加 url，如图 2-19 所示。

```
1   <!--pages/douban/index.wxml-->
2   <swiper style="height:450rpx" indicator-dots autoplay="true" interval="5000" duration="1000">
3     <swiper-item wx:for="{{ boards[0].movies }}" wx:key="{{ item.id }}">
4       <navigator url="item?id={{item.id}}" hover-class="none">
5         <image style="height:450rpx;width:750rpx;" src="{{ item.images.large }}" mode="aspectFill" />
6       </navigator>
7     </swiper-item>
8   </swiper>
```

图 2-19

此时，刷新项目，会发现效果已经完成。

2.5.5 下载源码

在本节的学习过程中，如果遇到问题，可以通过如下方式来解决。

- 加入小程序微信群与作者及其他读者一同探讨。微信扫描前言中的二维码，关注“艺述思维”，发送“小程序”进群。
- 下载作者的源码，对照查找问题。在公众号中发送“豆豆电影 2.5”获取下载地址。

2.6 实现电影列表页

本节将实现一个可以上下滚动的列表，当滚动到底部时提示“继续滑动加载更多”，如

图 2-20 所示。

2.6.1 使用 finally 方法

使用 2.1.3 节快速创建页面的方法，创建 pages/douban/list 页面，用于展示电影榜单列表。

在文档树中打开 pages/douban/list.js 文件，修改 data 变量为：

```
data: {
    type: 'in_theaters',
    page: 1,
    size: 20,
    total: 1,
    movies: []
}
```

其中，type 是调用豆瓣 API 时需要用到的电影类型。page 是分页的页码，代表当前是第几页，size 代表每次最多拉取多少条数据，total 需要从豆瓣服务器获取，默认设置为 1，movies 是拉取到的数据。由于允许查看多页内容，因此 movies 是一个累加数组。

图 2-20

然后，在页面数据变量 data 下方添加一个 retrieve 方法：

```
retrieve() {
    let app = getApp()
    let start = (this.data.page - 1) * this.data.size

    wx.showLoading({
        title: '加载中'
    })

    return app.request(`https://api.douban.com/v2/movie/${this.data.type}?start=
        ${start}&count=${this.data.size}`)
        .then(res => {
            if (res.subjects.length) {
                let movies = this.data.movies.concat(res.subjects)
                this.setData({ movies: movies, total: res.total })
                wx.setNavigationBarTitle({ title: res.title })
                console.log(movies)
            }
        }).catch(err => {
            console.error(err)
        }).finally( ()=> {
            wx.hideLoading()
        })
}
```

这个方法用于从豆瓣 API 分页拉取数据，默认是从第 1 页拉取。如果当前页数大于总页数，则不再拉取。当前页数的改变在另一个函数中，稍后会看到。

app.request 函数返回 Promise 对象，无论接口拉取失败还是成功，finally 回调都会执行，所以在这里需要隐藏加载提示。wx.hideLoading 是隐藏加载提示的界面交互 API，与之相对应的，先于 app.request 调用的 wx.showLoading 是显示加载提示的 API。

concat 是 js 数组的方法，用于连接两个数组，拼接元素并返回新数组。

console.log(movies) 这行代码用于在控制台窗口打印数据。

按 <Command+B> 组合键（或 <Ctrl+B> 组合键）刷新项目，或者等待工具自动刷新，在调试工具区域，查看 Console 面板，如图 2-21 所示，会看到数据输出。

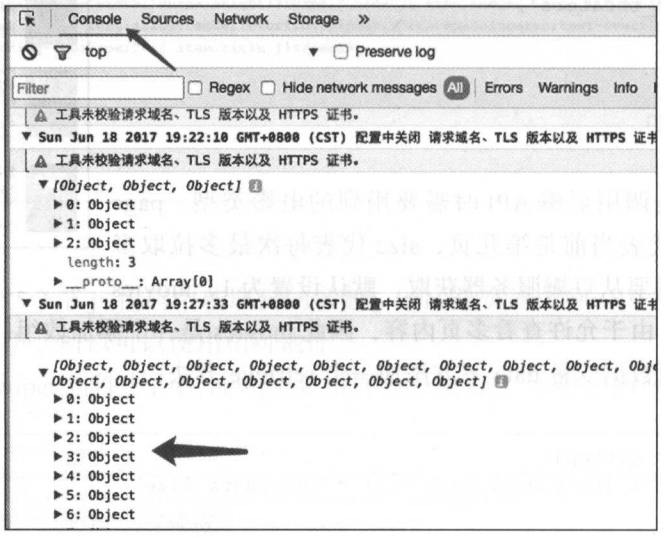


图 2-21

2.6.2 模板组件

在文档中选择 douban 文件夹并右击，从弹出的快捷菜单中选择“新建”命令，如图 2-22 所示。

选择新建文件的类型为 wxml，命名为 list-template.wxml。下面就在这个文件中新建一个模板组件，然后在 pages/douban/list.wxml 页面中使用。

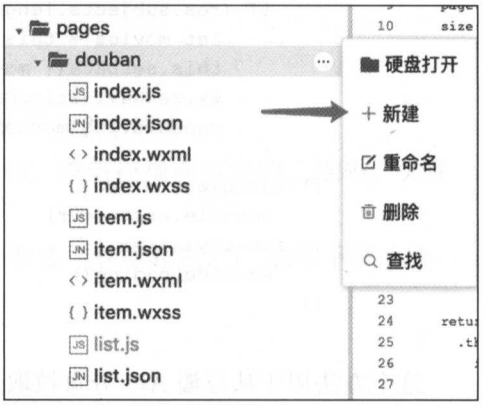


图 2-22

在文档树中打开 list-template.wxml 文件，修改 wxml 标签为：

```
<template name="list-template">
    <scroll-view enable-back-to-top scroll-y
        bindscrolltolower="loadMorePage">
        <view class="weui-panel">
            <view class="weui-panel__bd">
```

```
                <navigator wx:for="{{ movies }}" wx:key="{{ item.id }}" url="item?id=
                    {{ item.id }}" class="weui-media-box weui-media-box_appmsg" hover-
                    class= "weui-cell_active">
                    <view class="weui-media-box__hd weui-media-box__hd_in-appmsg"
                        style="height:inherit;width:120rpx">
                        <image style="width: 128rpx;height: 168rpx;" class="weui-
                            media-box__thumb" src="{{ item.images.small }}" />
                    </view>
                    <view class="weui-media-box__bd weui-media-box__bd_in-appmsg">
                        <view class="weui-media-box__title">{{ item.title }}</view>
                        <view class="weui-media-box__desc">{{ item.original_title }}
                            ({{ item.year }})</view>
                        <view class="weui-media-box__info">
                            导演: <block wx:for="{{ item.directors }}" wx:key="{{
                                item.id }}"> {{ item.name }} </block>
                        </view>
                    </view>
                    <view class="weui-media-box__ft">
                        <view class="weui-badge">{{ item.rating.average }}</view>
                    </view>
                </navigator>
            </view>
        </view>
        <view class="weui-loadmore" wx:if="{{total>page}}">
            <view class="weui-loadmore__tips">继续向下滑动加载更多内容</view>
        </view>
    </scroll-view>
</template>
```

在一个文件中定义模板组件，以便于在其他页面中复用该组件。不过，在这个项目中，只有一个页面用到了该组件，定义该模板组件的目的仅在于说明定义方法。以下是上述代码的说明。

- template 用于声明标签，name 用于指定模板名称，该名称将在 pages/douban/list.wxml 中使用。
- enable-back-to-top 属性用于实现单击标题栏回到顶部。
- bindscrolltolower 用于绑定滑动到底部的事件，loadMorePage 是一个尚未定义的方法。模板文件没有对应的 js 文件，稍后将在 pages/douban/list.js 中定义这个方法。

该模板组件中取用了三个模板变量：movies、page、total。它们将在模板被调用时传入。

在文档树中打开 pages/douban/list.wxml 文件，修改 wxml 标签为：

```
<import src="list-template"/>
<template is="list-template" data="{{ movies,total,page }}"/>
```

import 标签用于引入模板文件，src 属性设置的是相对地址，因为 list-template.wxml 文件与 list.wxml 文件位于同一目录之下。在小程序中，对于 wxml 文件的引用，都不带扩展名。

在 list-template.wxml 中，template 是定义标签，在 list.wxml 中是实例化标签，is 指示模板名称，data 是一个模板变量列表，是在模板中用到的变量，其中被传入的变量的顺序没有先后之分，“movies、total、page”与“total、page、movies”的效果是一样的。

2.6.3 加载更多

本节将实现向上滑动加载更多数据的功能。

实现加载更多的功能，需要用到 loadMorePage 方法，但在模拟文件 list-template.wxml 中使用的 loadMorePage 方法还没有被定义，因此在这个方法中需要将 page 加 1，然后再次拉取分页数据。

打开 pages/douban/list.js 文件，添加 loadMorePage 函数，代码如下：

```
loadMorePage(){
    if (this.data.page > this.data.total) return
    this.data.page++
    this.retrieve()
}
```

按 <Command+B> 组合键（或 <Ctrl+B> 组合键）测试一下，在列表页滑动页面至底部后继续向下滑动，发现并没有触发 loadMorePage 函数。这是为什么呢？

导航到列表页，在调试工具区打开 Wxml 面板，单击 scroll-view 这一行，如图 2-23 所示。

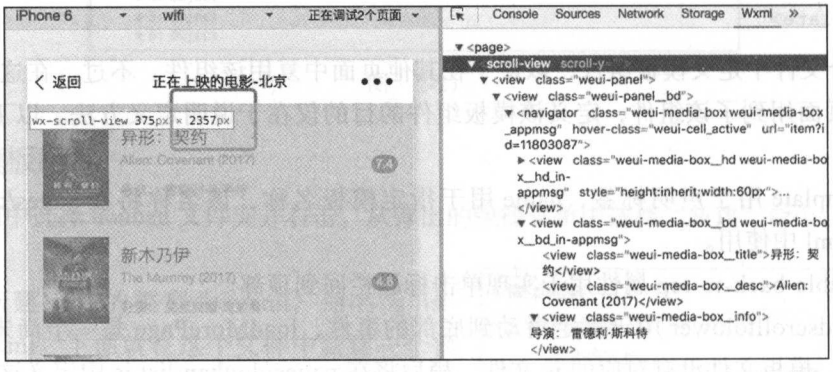


图 2-23

此时，会发现列表的高度是 2357px，这个高度明显超出了屏幕。这个高度是无法触发 scrolltolower 事件的，因为无法 scroll 到那里。再者，在 2000+ 这个高度，scroll-view 右侧没有出现滚动条，这是不正常的。我们猜想，此时的内容可以滚动，是因为 page 在滚动，并不是 scroll-view 在滚动。那么如何验证这个猜想呢？

打开 pages/douban/list.wxml 文件，在 import 这一行代码下面随意添加一些 UI，例如：

```
<view class="weui-loadmore">
```

```
    <view class="weui-loadmore__tips">测试滚动容器</view>
</view>
```

再次测试，向上滚动，会发现新添加的测试 UI 会随电影列表一直向上滚动，如图 2-24 所示。

图 2-24

这说明此时的滚动是 page 级别的滚动，scroll-view 没有发挥作用。

删除测试 UI，再在文档树中打开 pages/douban/list-template.wxml 文件，在 scroll-view 组件的 style 属性上添加内联样式，代码如下：

```
<scroll-view style="display:inline" enable-back-to-top scroll-y bindscrolltolower="loadMorePage">
...
```

scroll-view 默认的 display 样式是 block，它会让元素显示为块状元素，这就使得高度将按实际大小来显示，于是滚动失效。inline 代表将元素显示为内联元素，于是滚动条出现。

按 <Command+B> 组合键（或 <Ctrl+B> 组合键）刷新项目，会发现已经可以滚动，并能触发加载更多。

2.6.4 如何调试

到目前为止，已经实现了向上滚动加载更多的效果。但在测试中我们会发现存在如下两个问题。

- ❑ 一直可以触发加载更多，并且最后几页的数据已经重复。
- ❑ “继续滑动加载更多”的提示一直存在。

通过 console.log 打印获取到的数据如图 2-25 所示。

```
31          if (res.subjects.length) {
32            console.log(res)
33            let movies = this.data.movies.concat(res.subjects)
```

图 2-25

在 Console 面板中观察数据输出，如图 2-26 所示。

多次测试可以发现，豆瓣 API 中返回的 total 数据并非指总页数，而是指总条目数。

在文档树中打开 page/douban/list.js 文件，修改 retrieve 函数，如图 2-27 所示。

这样修改之后，就会先由总条目数和 size 计算一下总页数，然后再调用 setData。Math.floor 对小数点向下取整，返回不大于小数的最小整数。

再次测试，拉取重复数据的问题就解决了。

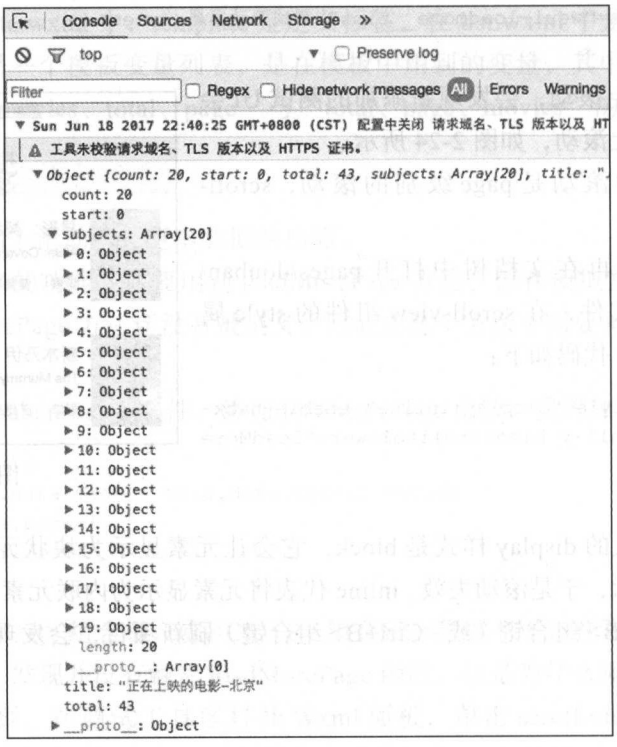


图 2-26

```
34          let total = Math.floor(res.total/this.data.size)
35          this.setData({ movies: movies, total: total })
```

图 2-27

2.6.5 刷新视图

但是“继续滑动加载更多”的提示仍然存在，这是由于绑定在模板组件 list-template 上的 page 变量没有刷新。代码如下：

```
this.data.page++
```

这样的代码只是递增 page 的数值，绑定在 list 页面上的 page 变量并没有更新。

将 retrieve 函数中的这行代码：

```
this.setData({ movies: movies, total: total})
```

修改为：

```
this.setData({ movies: movies, total: total, page: this.data.page })
```

修改之后，page 变量就会及时刷新，“继续滑动加载更多”的提示也不复存在。

setData 在设置变量的时候会通知视图刷新这些变量。在修改之前，虽然 total 被刷新

了，但是留在模板组件中的 page 变量一直是旧的数值 1，所以下面这段代码中的 if 判断没有如期发挥作用：

```
<view class="weui-loadmore" wx:if="{{total > page}}">
    <view class="weui-loadmore__tips">继续向上滑动加载更多内容</view>
</view>
```

2.6.6 下载源码

在本节的学习过程中，如果遇到问题，可以通过如下方式来解决。

- 加入小程序微信群与作者及其他读者一同探讨。微信扫描前言中的二维码，关注“艺述思维”，发送“小程序”进群。
- 下载作者的源码，对照查找问题。在公众号中发送“豆豆电影 2.6”获取下载地址。

2.7 实现下拉刷新功能

2.7.1 小程序中的下拉更新 API

如图 2-28 所示，在页面顶端下拉，就会自动刷新页面，本节将实现这个功能。

图 2-28

事实上，小程序本身已经提供了实现下拉更新的 API。

在文档树中打开 pages/douban/index.js 文件，添加一个 onPullDownRefresh 方法，代码如下：

```
onPullDownRefresh () {
    this.retrieveData().then(() => wx.stopPullDown-
        Refresh())
}
```

在这里使用箭头函数定义 then 方法需要的匿名函数，没有使用花括号 {} 将函数体括起来，是因为代码只有一行，省略了。

这里也体现了使用 Promise 编程的好处，因为 retrieveData 函数返回的是一个 Promise 对象，所以对它可以进行 then 调用。

在 onLoad 方法中，调用 retrieveData 之后就不需要再调用 wx.stopPullDownRefresh 方法了，只有在 onPullDownRefresh 函数中才需要。onPullDownRefresh 是小程序规定的下拉函数，在 Page 中定义它就意味着下拉更新功能实现了一半。

在文档树中打开 pages/douban/index.json 文件，json 文件是页面的配置文件。在花括号 {} 里输入“ en”，编程器会自动实时提示“ enablePullDownRefresh”，按 <Enter> 键，设置其值为 true。至此，下拉更新的功能已全部完成。

思考 当下拉更新时，如何显示一个“加载中”的提示，并在加载完成后自动隐藏提示。本节所附源码已实现了这个功能。

2.7.2 下载源码

在本节的学习过程中，如果遇到问题，可以通过如下方式来解决。

- 加入小程序微信群与作者及其他读者一同探讨。微信扫描前言中的二维码，关注“艺述思维”，发送“小程序”进群。
- 下载作者的源码，对照查找问题。在公众号中发送“豆豆电影 2.7”获取下载地址。

2.8 实现搜索功能

在首页单击搜索框进入搜索页面，输入要搜索的关键词，单击“搜索”按钮，展示的电影列表如图 2-29 所示；在搜索页面，提供即时匹配关键字的功能，如图 2-30 所示。本节将实现这个功能。

图 2-29

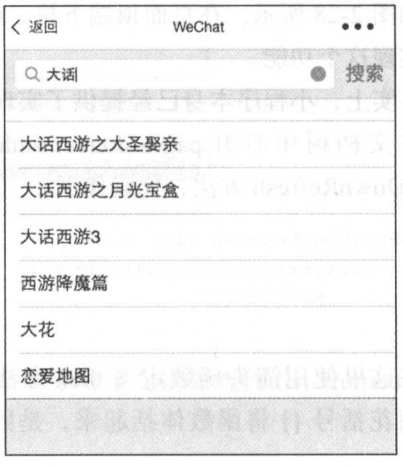


图 2-30

在文档树中打开 pages/douban/index.wxml 文件，在最下面添加如下代码：

```
<view class="weui-search-bar" style="position: absolute;top:0;width:100%;opacity: .8;">
    <navigator url="search" class="weui-search-bar__form">
        <view class="weui-search-bar__box">
            <icon class="weui-icon-search_in-box" type="search" size="14"></icon>
            <input type="text" class="weui-search-bar__input" />
        </view>
        <label class="weui-search-bar__label">
            <icon class="weui-icon-search" type="search" size="14"></icon>
```

```
            <view class="weui-search-bar__text">搜索</view>
        </label>
    </navigator>
</view>
```

这部分 wxml 标签代码，在首页定义了一个八分透明的“假”的搜索框，效果如图 2-31 所示。

图 2-31

单击搜索框时，会转到 search 页面（这个页面目前尚未定义）。

在文档树中打开 app.json，使用 2.1.3 节的快捷方法创建 pages/douban/search 页面。

打开 search.wxml 页面，修改 wxml 标签代码为：

```
<view class="weui-search-bar">
    <view class="weui-search-bar__form">
        <view class="weui-search-bar__box">
            <icon class="weui-icon-search_in-box" type="search" size="14"></icon>
            <input type="text" class="weui-search-bar__input" value= {{searchWords}}"
                focus="{{searchInputFocus}}" bindinput="onSearchInputType" />
            <!--清空内容的icon-->
            <view class="weui-icon-clear" wx:if="{{searchWords.length > 0}}"
                bindtap="clearSearchInput">
                <icon type="clear" size="14"></icon>
            </view>
        </view>
        <label class="weui-search-bar__label" hidden="{{searchInputFocus}}"
            bindtap="showSearchInput">
            <icon class="weui-icon-search" type="search" size="14"></icon>
            <view class="weui-search-bar__text">搜索</view>
        </label>
        </view>
        <view class="weui-search-bar__cancel-btn" hidden="{{!searchInputFocus}}"
            bindtap="onTapSearchBtn">
            <block wx:if="{{searchWords.length == 0}}">取消</block>
            <block wx:else>搜索</block>
        </view>
    </view>
    <!--即时搜索词列表-->
    <view class="weui-cells searchbar-result" wx:if="{{wordsList.length > 0}}">
        <navigator url="item?id={{item.id}}" wx:for="{{wordsList}}"
            wx:key="{{item.id}}" class="weui-cell" hover-class="weui-cell_active">
            <view class="weui-cell__bd">
                <view>{{item.title}}</view>
            </view>
        </navigator>
    </view>

<import src="list-template" />
<template is="list-template" data="{{ movies,total,page }}" />
```

打开 pages/douban/search.js 文件，将代码修改为：

```
Page({
    data: {
        searchInputFocus: true,
        searchWords: "",
        wordsList: [],
        size: 20,
        page: 1,
        movies: [],
        requestInternal:-1,
    },
    onTapSearchBtn() {
        console.log("words",this.data.searchWords)
        if (this.data.searchWords != "") {
            this.retrieve()
        }
        this.setData({
            searchInputFocus: false,
            searchWords: "",
            wordsList: []
        });
    },
    retrieve() {
        let app = getApp()
        let start = (this.data.page - 1) * this.data.size
        wx.showLoading({ title: '加载中' })

        return app.request(`https://api.douban.com/v2/movie/search?q=${this.data.
            searchWords}&start=${start}&count=${this.data.size}`)
            .then(res => {
                console.log("res",res)
                if (res.subjects.length) {
                    let movies = this.data.movies.concat(res.subjects)
                    let total = Math.floor(res.total / this.data.size)
                    this.setData({ movies: movies, total: total, page: this.data.
                        page,wordsList:[] })
                    wx.setNavigationBarTitle({ title: res.title })
                }
            }).catch(err => {
                console.error(err)
            }).finally(() => {
                wx.hideLoading()
            })
        },
        showSearchInput(){
            this.setData({
                searchInputFocus: true
        });
    },
    // 清空输入框内容
```

```
    clearSearchInput() {
        this.setData({
            searchWords: ""
        });
    },
    // 当在搜索框输入内容
    onSearchInputType(e) {
        let app = getApp()
        let words = e.detail.value
        this.setData({
            searchWords: words
        });
        clearTimeout(this.data.requestInternal)
        this.data.requestInternal = setTimeout(()=>{

            app.request(`https://api.douban.com/v2/movie/search?q=${words}&start
                =0&count=6`).then(d => {
                console.log(d)
                if (d.subjects.length) {
                    this.setData({
                        wordsList: d.subjects
                    });
                }
            })
        },2000)
    }
})
```

在上述代码中，searchInputFocus 控制 input 组件的焦点，当其为 true 时，input 组件自动聚集。

bindinput 用于绑定输入事件，在 onSearchInputType 函数中，先将输入的文本从 e.detail.value 中取出并保存。为了避免在用户尚未完成输入时频繁调用，可使用 setTimeout 设置在 2 秒之后调用豆瓣接口。在下次未调用之前，如果有新的输入，则使用 clearTimeout 清除上一次的间隔调用。

注意 setTimeout() 只执行一次 code。如果要多次调用，请使用 setInterval() 或者让 code 自身再次调用 setTimeout()。

对于下面这段代码：

```
<import src="list-template" />
<template is="list-template" data="{{ movies,total,page }}" />
```

其内容与 pages/boudan/list.wxml 相同，表示在这里复用了模板组件 list-template，这就是定义模板组件的好处。search.js 文件中的 retrieve 函数与 pages/douban/list.js 文件中的 retrieve 函数类似，在开发中前者可以基于后者复制修改。由于接口调用代码与 setData 代码混合在一起了，因此不适合进行进一步的拆分抽象。

单击输入框表象 UI 时，触发 showSearchInput 函数，在这个函数中仅设置焦点变量 searchInputFocus 为 true。

单击“x”号图标时，触发 clearSearchInput 函数，设置 searchWords 为空。当 searchWords 为空时，“x”号图标将不再显示。

```
<view class="weui-search-bar__cancel-btn" hidden="{{!searchInputFocus}}" bindtap=
    "onTapSearchBtn">
    <block wx:if="{{searchWords.length == 0}}">取消</block>
    <block wx:else>搜索</block>
</view>
```

以上代码使用条件绑定，当 searchWords 的字符串长度不为零时，显示“取消”，否则显示“搜索”。

这里使用 bindtap 绑定 onTapSearchBtn 函数。在 onTapSearchBtn 函数中，先清除搜索变量，如果搜索词不为空，则调用 retrieve 函数。

在本节的学习过程中，如果遇到问题，可以通过如下方式来解决。

- 加入小程序微信群与作者及其他读者一同探讨。扫描前言中的二维码，关注“艺述思维”，发送“小程序”进群。
- 下载作者的源码，对照查找问题。在公众号中发送“豆豆电影 2.8”获取下载地址。

2.9 提交

本节主要设置作品名称和头像，然后配置服务器域名并提交审核。

2.9.1 修改信息

在文档树中打开 app.json 文件，找到 navigationBarTitleText 字段，将其值修改为“豆豆电影”。

在电脑上登录 https://mp.weixin.qq.com，打开“设置”→“基本设置”，在这里修改小程序名称、小程序头像和介绍。需要扫码认证管理员，即注册时设置的微信账号。

例如将介绍修改为“分享最新的电影资讯”后，单击“确定”按钮，提交审核，如图 2-32 所示。

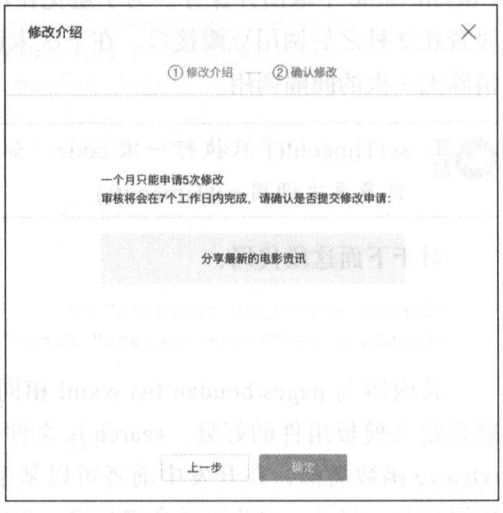


图 2-32

审核会有一段时间的审核期，一般是在 3 天之内完成。由微信之外的公司负责审核，审核人员的上班时间是周一至周五。

2.9.2 使用 Sketch 生成头像

修改头像需要再次验证管理员权限，头像的大小最好为 144x144 像素。

到 http://www.sketchcn.com/ 上下载 Sketch 软件。Sketch 软件是一款轻量、易用的矢量图设计工具，设计 Logo、图标比 Photoshop 更方便。

打开 Sketch 软件，新建一个 Document，在菜单中选择 Insert → Shape → Rectangle 命令，如图 2-33 所示。

然后新建一个矩形，在右边的属性面板中设置宽、高为 144px，设置 Fill 颜色为 #ff00aa。

接着，在菜单栏中选择 Insert → Text 命令，插入一个文本“豆豆电影”，调整字体、大小、间距和位置，如图 2-34 所示。

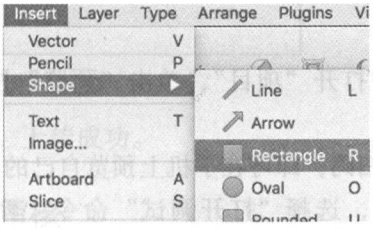


图 2-33

图 2-34

将文本与图形群组，在右边的导出面板中，按默认 1 倍的大小导出 png 图片。

现在回到小程序微信后台，将新导出的头像上传。圆形头像是在方形的基础上自动生成的，所以在上面的制作中，需要加一个边距，如图 2-35 所示。

图 2-35

这样就完成了头像修改的操作。

2.9.3 配置域名器域名

在微信小程序后台，打开“设置”→“开发设置”，找到“服务器域名”，单击“开始配置”按钮，在“request 合法域名”文本框中填写“http://api.douban.com”，如图 2-36 所示。

然后，单击“保存并提交”按钮。

在小程序代码中，我们仅调用过 api.douban.com 这一个域名。在设置服务器域名之后，在微信 Web 开发者工具中，即可打开服务器域名验证，但此举没有必要。

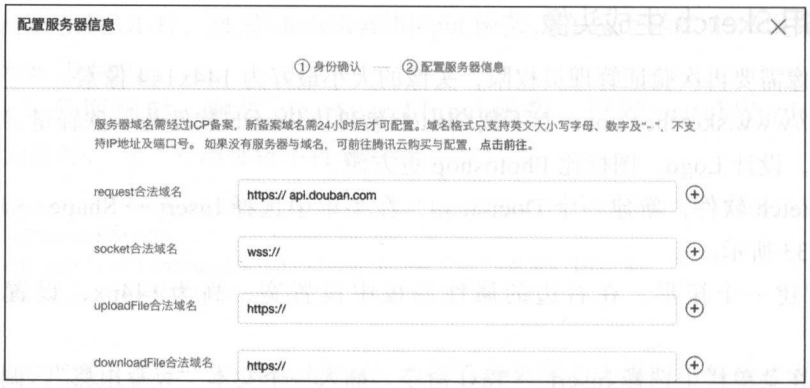


图 2-36

2.9.4 在手机上预览

到微信开发者工具中，从左边工具栏中打开“项目”，单击“预览”按钮，如图 2-37 所示。

打开小程序预览面板，使用管理员微信扫码，即可在手机上预览自己的作品。

在打开的微信小程序的右上角菜单中，选择“打开调试”命令，可在右下角开启 vConsole 面板，查看 Console 输出，如图 2-38 所示。

图 2-37

图 2-38

2.9.5 上传版本

在微信开发者工具中，从左边工具栏中打开“项目”，单击“上传”按钮，如图 2-39 所示。

然后在版本上传操作面板中填写版本号（即 0.1），单击“上传”按钮，如图 2-40 所示。

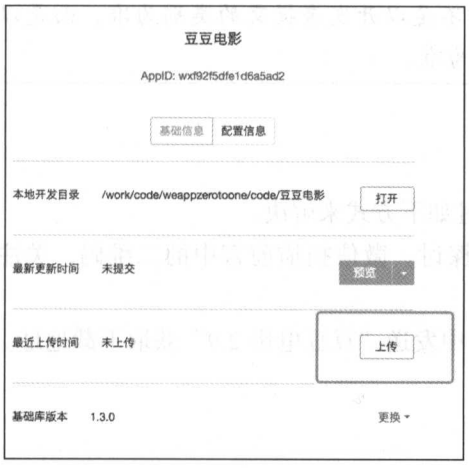


图 2-39

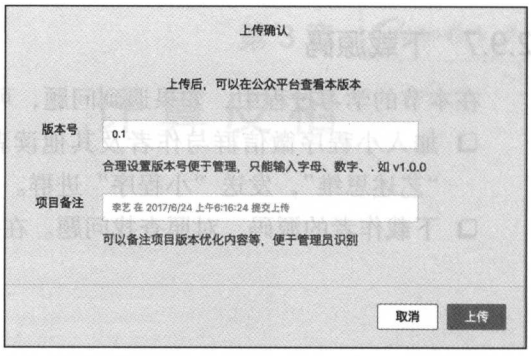


图 2-40

稍等片刻，上传成功。

2.9.6 提交审核

回到微信小程序后台，选择“开发管理”，即可看到上传的版本，如图 2-41 所示。

单击“提交审核”按钮，同意审核规则，验证管理员权限，打开审核信息面板，即可得到图 2-42 所示的界面。

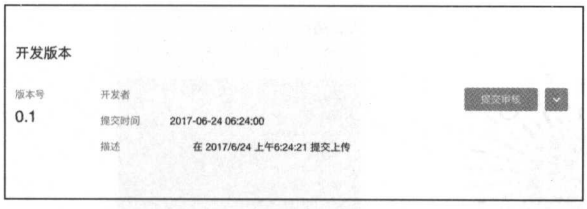


图 2-41

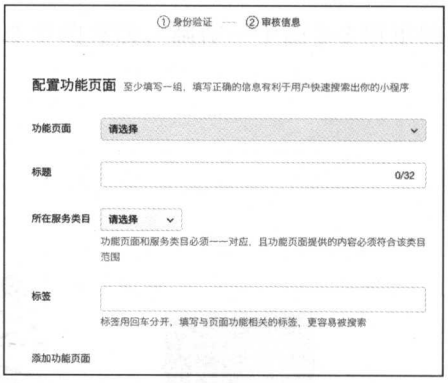


图 2-42

在“功能页面”选择“ pages/douban/index”，在“标题”文本框中输入“最新电影”，选择所在类目，在“标签”文本框中输入“电影”，然后按 <Enter> 键，单击“提交审核”按钮。

在审核信息的填写面板中，可以单击“添加功能页面”，添加其他页面的描述。

审核是由微信之外的第三方团队来负责的，一般 1~3 天便可审核通过。

电影属于文娱 – 资讯类别，截至本书写作之时，这一类别尚未对个人小程序账号开放，不会通过审核。微信小程序的类别审核不是以开发者提交的类别为准，而是以审核人员通过名称、简介、内容判断的类别为准。

2.9.7 下载源码

在本节的学习过程中，如果遇到问题，可以通过如下方式来解决。

- 加入小程序微信群与作者及其他读者一同探讨。微信扫描前言中的二维码，关注“艺述思维”，发送“小程序”进群。
- 下载作者的源码，对照查找问题。在公众号中发送“豆豆电影 2.9”获取下载地址。

第 3 章 Chapter 3

计算皮相

本章将实现一个简单的标准计算器，这个计算器称为“计算皮相”，因为它本身并不像真正的计算器那样具有计算的功能，只是提供了一个输入界面（如图 3-1 所示），真正的计算其实是由计算机或手机来完成的。

可用微信扫描图 3-2 所示的小程序码，在线体验。

按照 1.1 节介绍的方法，前往 https://mp.weixin.qq.com 注册一个全新的小程序账号，将其名称修改为“计算皮相”，将介绍修改为“一个可以在群内分享的简约计算器”，然后将服务类目修改为“工具”→“效率”。最后使用 2.9.2 节的方法，制作一个图 3-3 所示的图像，将其作为“计算皮相”的头像。

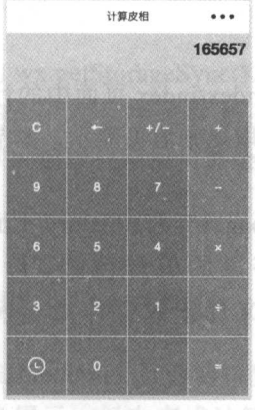


图 3-1

图 3-2

图 3-3

3.1 使用模板创建项目

打开微信开发者工具，新建项目，在“项目目录”中选择一个空目录，并且反选“在当前目录中创建 quick start 项目”，如图 3-4 所示。

图 3-4

不过，空项目并不会创建任何文件。

之后，参照 2.1.4 节的方法，下载 sim.js 源码，并将之放在“计算皮相”项目的根目录之下，将 sim.js/template 目录下的所有文件、目录移至“计算皮相”的根目录下。template 是引用 sim.js 类库的默认结构，直接复制过来可免去手动创建的麻烦，同时也不必创建 quick start 项目中其他多余的文件。

3.2 实现 history 页面

pages/index/history 页面将显示计算过的历史，pages/index/index 页面中的计算记录将被保存到本地缓存中，然后在 pages/index/history 页面被提取出来渲染展示，本节将来实现这个功能。

在文档树中打开 app.json，使用 2.1.3 节中的方法新建 pages/index/history 页面。

在 app.json 文件中，将“navigationBarTitleText”字段的值修改为“计算皮相”，navigationBarTitleText 代表小程序的标题。如果页配置文件中没有指定 navigationBarTitleText，则默认使用 app.json 中的全局设置。

打开 pages/index/history.wxml，以如下代码替换之：

```
<view class="weui-cells">
    <view class="weui-cell" wx:for="{{logs}}" wx:key="*this">
```

```
        <view class="weui-cell__bd">{{item}}</view>
    </view>
</view>
```

在上述代码中，wx:for 是列表渲染标签，默认当前循环项的变量名为 item，所以可以直接使用 {{item}} 来绑定。wx:key 用于在动态列表渲染中保持子项的特征和状态，指定 wx:key 可以提高渲染效率，不指定则会收到一个警告，如图 3-5 所示。

```
⚠ Now you can provide attr "wx:key" for a "wx:for" to improve performance.
  1 | <!--pages/index/history.wxml-->
  2 | <view class="weui-cells">
> 3 |   <view class="weui-cell" wx:for="{{logs}}">
    |     ^
  4 |     <view class="weui-cell__bd">{{item}}</view>
  5 |   </view>
  6 | </view>
```

图 3-5

使用 wx.getStorageSync 接口

wx.GetStorageSync 接口与 2.3.3 节使用过的 wx.GetStorage 接口的作用相同，不同之处在于，前者是同步调用，后者是异步调用；前者调用之后，可立即返回结果，后者则需要在回调函数中获取结果。

打开 pages/index/history.js 文件，将页面的初始数据 data 修改为：

```
data:{
    logs:[]
}
```

其中，logs 用于记录计算历史。

然后，将 onLoad 函数替换为如下代码：

```
onLoad(options){
    var logs = wx.getStorageSync('calclogs');
    this.setData({"logs":logs});
}
```

这里的 wx.getStorageSync(Key) 是小程序同步缓存 API，即从本地缓存中同步获取 Key 指定的内容。凡接口中带有 Sync 字样的，均是同步 API，使用同步 API 可以避免写匿名回调的麻烦。但缓存获取可能会失败，一般情况下会将同步代码放在 try catch 代码中捕捉异常。

3.3 实现 index 主页

pages/index/index 页面将放置计算器按钮，并在逻辑层完成主要的计算逻辑，然后在视图层渲染展示。每次完成计算之后，都需要将记录存进本地缓存，以便在 pages/index/history 页面取用。本节将主要完成这些功能。

在文档树中打开 pages/index/index.wxml 文件，将其中的内容替换为如下标签代码：

```
<view style="background-color:#d8d8d8;height:100%;">
    <view style="text-align:right;width:100%;padding:10rpx;font-weight:bold;font-
     size:50rpx;box-sizing:border-box;border-top:1px solid #eee;border-top:1px
     solid #eee;height:200rpx;word-wrap:break-word; word-break:normal;">
      {{screenData}}
    </view>
        <view bindtap="tapSymbolBtn" style="width: 100%; color: #fff;">
            <view class="row">
            <view class="orange" data-symbol="clear">C</view>
            <view class="orange" data-symbol="back">←</view>
            <view class="orange" data-symbol="toggle">+/-</view>
            <view class="orange" data-symbol="+">+</view>
        </view>
        <view class="row">
            <view class="blue" data-symbol="9">9</view>
            <view class="blue" data-symbol="8">8</view>
            <view class="blue" data-symbol="7">7</view>
            <view class="orange" data-symbol="-">-</view>
        </view>
        <view class="row">
            <view class="blue" data-symbol="6">6</view>
            <view class="blue" data-symbol="5">5</view>
            <view class="blue" data-symbol="4">4</view>
            <view class="orange" data-symbol="*">×</view>
        </view>
        <view class="row">
            <view class="blue" data-symbol="3">3</view>
            <view class="blue" data-symbol="2">2</view>
            <view class="blue" data-symbol="1">1</view>
            <view class="orange" data-symbol="÷">÷</view>
        </view>
        <view class="row">
            <navigator url="history" class="blue">
                <icon type="waiting_circle" color="white"
                  size="25" />
            </navigator>
            <view class="blue" data-symbol="0">0</view>
            <view class="blue" data-symbol=".">.</view>
            <view class="orange" data-symbol="=">=</view>
        </view>
    </view>
</view>
```

该文件将用于构建计算器的 UI，如图 3-6 所示。

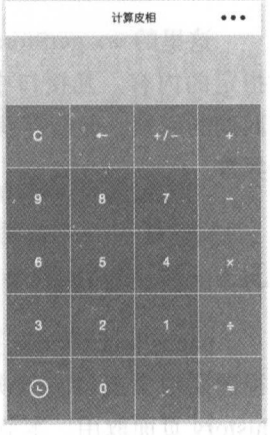


图 3-6

在上述代码中，screenData 用于绑定计算结果，由于计算结果可能过长，所以使用了自动换行的内嵌样式，代码如下：

```
word-wrap:break-word; word-break:normal;
```

3.3.1 冒泡事件

在前面给出的代码中，bindtap="tapSymbolBtn" 用于将 tap 事件绑定到 tapSymbolBtn 函数中。使用 bind 方式绑定的事件具有冒泡属性，虽然事件绑定在外围 view 容器之上，但其内部所有 blue 或 orange 按钮的单击事件都会触发 tapSymbolBtn 函数。

在标签定义中，view 基本上是万能 UI 组件。这个计算器表象也是以 view 嵌套定义的。class 用于指定组件的样式名，这与 html 标签一致。而 row、blue、orange 这些则都是在 pages/index/index.wxss 中定义的样式。

icon 是图标组件，有效的 type 属性值包括 success、info、warn、waiting、cancel、download、search、clear 等。这些 type 名称可以与“_no_circle”或“_circle”相结合组合成新的 type。官方文档并没有详细列举每一个有效的组合 type 值，并不是每一个组合都是有效的，具体以实践测试为准。

“data-symbol”用于在组件上定义一个名为“symbol”的自定义属性，使用自定义属性，可以将视图层的数据传至代码层。在事件函数中，可以使用 event.target.dataset 取得事件源组件上定义的所有自定义属性的集合。

最后，因为 history 页面与 index 页面同级，位于同一目录之下，所以这里的 navigator 组件，其 url 属性可以简写为“history”，也就是说，使用的是相对地址。

3.3.2 样式选择器

在文档树中打开 pages/index/index.wxss 文件，添加如下样式声明：

```
.row {
    display: flex;
    flex-direction: row;
    flex: 1;
    width: 100%;
    height: 185rpx;
    background-color: #eee;
}
.row view,.row navigator {
    width:25%;
    display: flex;
    align-items: center;
    flex-direction: column;
    justify-content: center;
    margin-top: 1px;
    margin-right: 1px;
}
.row view:active {
    background-color: #ff0000;
}
.orange {
    background: #f77d11;
```

```
}
.blue {
    background-color: #0099cc;
}
```

其中，样式声明“.row view:active”是用于单击按钮时的样式。单击计算器按钮时，其背景高亮反红就是由这个样式来实现的。

.row 是类选择器名称，用在组件的 class 属性中。“.row view”代表 class 属性，可针对 .row 内部的 view 组件定义它们应该具备的样式。view 是元素选择器。在 wxss 中使用元素选择器，可以免去多次设置 class 属性的麻烦。“.row navigator”样式声明与此同理。

在两个样式声明中间加“,”，代表并列，具有相同的样式，如“.row view,.row navigator”。

wxss 文件是样式文件。在小程序中有两种样式文件，一种是全局的 app.wxss，位于根目录之下；另一种页面的样式文件，与 wxml 文件同名并且位于同一目录下。

3.3.3 实现计算的逻辑

在文档树中打开 pages/index/index.js 文件，将页面初始数据 data 替换为：

```
data: {
    screenData: "0",
    lastIsOptSymbol: false,
    arr: [],
    logs: []
}
```

在上述代码中，screenData 用于记录输入和计算的结果。lastIsOptSymbol 代表上一次用户输入的字符是不是一个操作符。arr 是记录的输入队列，真正的计算依赖于它。logs 是计算素材与结果的历史记录，与 pages/index/history.js 中的 logs 其数据是相同的。

然后，在 pages/index/index.js 文件中依次添加如下函数：

```
// 设置屏显文本，如果没有文本或文本无效，则默认显示“0”
    setScreenData(data) {
        if (data == "" || data == "—") {
            data = 0;
        }
        this.setData({ "screenData": data });
    },
    // 退格操作
    backOperation() {
        if (this.data.arr.length == 0) {
            return
        }
        this.data.arr.pop();
        var data = this.data.screenData;
        data = data.substring(0, data.length - 1);
```

```
    this.setScreenData(data)
},
// 正负号操作
minusOperation() {
    if (this.data.arr.length == 0) {
        return
    }
    var data = this.data.screenData;
    var firstChar = data.charAt(0);
    // 反转正、负号
    if (firstChar == "—") {
        data = data.substr(1);
        //首位抽出
        this.data.arr.shift();
    } else {
        data = "—" + data;
        //首位推入
        this.data.arr.unshift("—");
    }
    this.setScreenData(data)
},
// 等于操作
equalOperation() {
    if (this.data.arr.length == 0) {
        return
    }
    var data = this.data.screenData;
    // 判断最后一个字符，如果不是数字，则返回
    var lastChar = data.charAt(data.length - 1);
    if (isNaN(lastChar)) {
        return;
    }

    var num = "";
    var lastOperator = "";
    var arr = this.data.arr;
    var optarr = [];

    for (var i in arr) {
        if (isNaN(arr[i]) == false || arr[i] == "." || arr[i] == "toggle") {
            // 如果是数字，或者“.”、“-”，则累加进num字符串
            num += arr[i];
        } else {
            // 剩下的是+-*/四则运算符
            lastOperator = arr[i];
            // 先推入数字
            optarr.push(num);
            // 再推入操作符
            optarr.push(arr[i]);
            // 清空num字符串，在下次循环中使用
            num = "";
```

```
        }
    }
    if (num) {
        optarr.push(Number(num))
    }

    var result = Number(optarr[0])
    // console.log(result);
    for (var i = 1; i < optarr.length; i = i + 2) {
        if (optarr[i] == "+") {
            // 加
            result += Number(optarr[i + 1]);
        } else if (optarr[i] == "-") {
            // 减
            result -= Number(optarr[i + 1]);
        } else if (optarr[i] == "*") {
            // 乘
            result *= Number(optarr[i + 1]);
        } else if (optarr[i] == "÷") {
            // 除
            result /= Number(optarr[i + 1]);
        }
    }
    // 存储历史记录
    this.data.logs.push(data + " = " + result);
    wx.setStorageSync("calclogs", this.data.logs);

    // 将本次计算结果存储进arr数组，以备下次计算
    this.data.arr.length = 0;
    this.data.arr.push(result);

    this.setScreenData(result + "")
},
numAndOtherOperation(symbol) {
    const numOperateSymbols = { "+": "+", "—": "-", "×": "*", "÷": "/", ".":
        "." }
    if (numOperateSymbols[symbol]) { //如果是运算符号+-*/.
        // 如果上次输入了运算符号，则返回，避免重复单击运算符号
        if (this.data.lastIsOptSymbol || this.data.screenData == 0) {
            return;
        }
    }

    var sd = this.data.screenData;
    var data;
    if (sd == 0) {
        data = symbol;
    } else {
        data = sd + symbol;
```

```
        }
        this.data.arr.push(symbol);
        this.setScreenData(data)

        this.data.lastIsOptSymbol = numOperateSymbols[symbol]
    },
    tapSymbolBtn(e) {
        var symbol = e.target.dataset.symbol;
        if (undefined == symbol) {
            return
        }
        switch (symbol) {
            case "back":
                //退格←
                this.backOperation()
                break
            case "clear":
                //清屏
                this.setScreenData("")
                this.data.arr.length = 0;
                break
            case "toggle":
                //正负号+/-
                this.minusOperation()
                break
            case "=":
                //等于=
                this.equalOperation()
                break
            default:
                this.numAndOtherOperation(symbol)
        }
    }
```

下面就来针对代码里的几个关键字进行说明。

(1) js 关键字 undefined

tapSymbolBtn 是在 index.wxml 文件中绑定的 tap 事件函数。e.target.dataset.symbol 用于获取自定义属性 data-symbol。并不是所有按钮都自定义了该属性，所以将它的值与 undefined 进行比对，如果未定义，则返回，代码如下：

```
if (undefined == symbol) {
    return
}
```

undefined 是 js 保留的关键字，代表变量指向了不存在的地址。

计算器主要有五类操作，即退格、清屏、正负号切换、等于计算、数字输入及其他操作。每一种操作除了清屏之外，均对应于一个独立的分支函数，这样比较便于代码的阅读和维护。

在清屏代码中，“this.data.arr.length = 0”这行代码通过将 length 设置为 0 来将数组 arr 清空，这种方式与“this.data.arr = []”等效，前者不用新建对象，效率比后者略高，是社区推荐的非标准方法。

（2）js 函数 chatAt

在函数 minusOperation 中，先使用“this.data.arr.length == 0”判断是否有数字输入，如果没有，则直接返回。

data.charAt(0) 用于提取字符串 data 的首位字符。

data.substr(1) 用于从位置 1 截断字符串并返回。substr 与 substring 实现了相同的功能，只是参数不同。

（3）js 函数 shift、unshift

this.data.arr.shift() 用于从数组 arr 的首位抽出一个元素，而 this.data.arr.unshift("-") 则是向数组 arr 推入一个字符“-”。

（4）js 函数 substring、substr

setScreenData 函数用于调用 setData 设置 screenData 变量，并在设置之前进行相关的有效性检查或预设。因为要多次使用这个逻辑，所以其被抽象抽取为一个独立的函数。

backOperation 函数的逻辑相对简单。this.data.arr.pop() 用于将数组 arr 的顶端，即最后一个推入的元素抛弃。

data.substring(0, data.length - 1) 用于截取出最后一个字符的所有字符。它与 substr 的不同之处在于，substring 接受的参数是起始位置，substr 则是开始位置和长度。

（5）js 函数 isNaN

equalOperation 是相对比较复杂的函数。isNaN(lastChar) 用于判断 lastChat 是不是非数字，如果是，则返回。

接下来使用一个 for 循环，将数组 arr 中的元素分组存入数组 optarr 中，并将相临的数字及小数点归在一起作为一个元素，而把操作符诸如“+-*/”等作为一个元素，为进行下一步操作做准备。

注意 如果在 PC 浏览器中，对于这样的一个字符串算式“12+3-5”，可以直接用 eval 函数计算结果，但在小程序中是不可以的。原因在于 eval 函数的宿主是 window 对象，而小程序没有 window 对象，也就没有 eval 函数。

代码 Number(optarr[0]) 可将整型数字转换为小数，String(result) 可将数字转换为字符串。

3.3.4 使用 wx.setStorageSync 接口

代码 wx.setStorageSync("calclogs", this.data.logs) 用于将最新的 logs 数组同步存入本地

缓存中，以便在 pages/index/history.js 中取用。

在函数 numAndOtherOperation 中，下面这行代码：

```
const numOperateSymbols = { "+": "+", "−": "-", "×": "*", "÷": "/", ".": "." }
```

定义了一个 numOperateSymbols 常量，该常量是一个集合对象。由于部分键值对名称与值不同，所以采用集合对象，而不用数组。

3.3.5 下载源码

在本节学习的过程中，如果遇到问题，可以通过如下方式来解决。

- 加入小程序微信群与作者及其他读者一同探讨。微信扫描前言中的二维码，关注“艺述思维”，发送“小程序”进群。
- 下载作者的源码，对照查找问题。在公众号发送“计算皮相 3.3”获取下载地址。

3.4 服务类目

若将上面的小程序内容按 2.9.6 节的方法提交审核，1～3 天后会收到微信公众平台的反馈，审核失败。原因是在 3.1 节我们选择的服务类目为“工具 - 效率”，微信建议的类目为“工具 - 计算类”。

见微信小程序审核规则第 2.1 条：

> 小程序的类目要和自身所提供的服务一致
>
> 1）小程序服务类目所对应的页面中的核心内容必须与该类目一致。
>
> 2）必须保证用户在该页面能够使用该服务类目，不得隐藏，不得进行多次跳转。

关于小程序的介绍也要如实、详尽地描述小程序的功能，否则审核不通过。例如这个小程序，如果将介绍笼统地写成“生活小工具”，则不被通过。

3.5 发布

如果小程序符合微信的审核规范，则在 1～3 天内会在管理员微信上收到审核通过的通知，如图 3-7 所示。

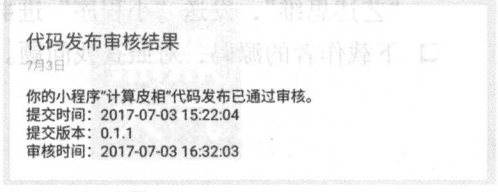


图 3-7

这时，使用小程序账号登录微信小程序后台，打开“开发管理”，单击“提交发布”按钮发布版本，如图 3-8 所示。

发布之后，在“设置”→“基本设置”页面，下载小程序码。本章示例的小程序码如

图 3-9 所示。

图 3-8

图 3-9

3.6 添加分享

在文档树中打开 pages/index/index.js 文件，在 data 数据下方添加 onShareApp-Message 函数，代码如下所示：

```
onShareAppMessage(res) {
    return {
        title: '可以分享计算结果的简约计算器',
        path: '/pages/index/index',
    }
}
```

title 是分享标题，path 是分享路径。普通的网页分享不带状态，小程序的分享则是带状态的，所以本示例的标题是“可以分享计算结果的简约计算器”。

微信规定，Page 内名为 onShareAppMessage 的函数用于转发消息，只有定义了该函数，小程序右上角的菜单才会出现分享按钮，如图 3-10 所示。

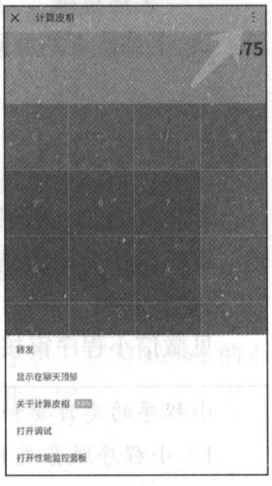


图 3-10

添加分享之后，需要重新上传版本、提交审核。

3.7 下载源码

在本节的学习过程中，如果遇到问题，可以通过如下方式来解决。

- 加入小程序微信群与作者及其他读者一同探讨。微信扫描前言中的二维码，关注“艺述思维”，发送“小程序”进群。
- 下载作者的源码，对照查找问题。在公众号中发送“计算皮相 3.6”获取下载地址。

第 4 章 Chapter 4

黑黑天气

本章将实现一个简单的天气小程序——黑黑天气（如图 4-1 所示）。用户可通过调用公共 API 接口，查询所在城市的天气状况。

前往 https://mp.weixin.qq.com，按照 1.1 节的方法，注册小程序账号。将名称修改为“黑黑天气”，将介绍修改为“简单个性天气小工具”。然后按照 2.9.2 节的方法，制作图 4-2 所示的头像，并进行修改。

图 4-1

图 4-2

打开微信开发者工具，按照 3.1 节的方法，基于 sim.js 项目模板，创建初始项目。

4.1 实现视图层

本节将主要实现 pages/index/index 页面的视图层，为了使小程序在不同设备上能有相同的 UI 效果，这里使用了响应式设计单位 rpx。

4.1.1 关于 rpx

rpx 是微信小程序中 css 的尺寸单位，可以根据屏幕宽度进行自适应。

在文档树中打开 pages/index/index.wxml，将内容替换为以下代码：

```
<!--pages/index/index.wxml-->
<view class="container">
    <view style="box-sizing: border-box;padding:10px;background-repeat: no-
        repeat;background-size: 100% 100%;background-image: url('/static/image/
        background/{{weather.today.typeBackgorund}}.jpg');width:750rpx;position:
        absolute;bottom:480rpx;top:0;color:#fff;">
        <view class="weui-flex">
            <view class="weui-flex__item" style="font-size:50rpx;color:#fff;">{{
                weather.today.wendu}}</view>
            <view class="center"><image style="width:50rpx;height:50rpx;"
                bindtap="loadWeather" src="/static/image/refesh.png"></image>
                </view>
        </view>

        <view class="weui-flex__item center">
            <view class="weui-grid" style="width: auto;border-style: none;">
                <image style="width: 320rpx;height: 280rpx;" src="/static/image/
                    white/{{weather.today.type}}.png"></image>
                <view class="weui-grid__label" style="color:#fff;font-size:
                    40rpx;">{{weather.today.type}}</view>
            </view>
        </view>

        <view class="weui-flex" style="display: flex;align-items: center">
            <view style="font-size:120rpx" class="weui-flex__item">
                {{weather.wendu}}° C
            </view>
            <view style="text-align: center" class="weui-flex__item">
                {{weather.today.week}} {{weather.city}}
            </view>
        </view>
    </view>
    <!--未来天气列表-->
    <view style="background: #fff;position: absolute;bottom: 0;width:750rpx;
        height: 480rpx;">
        <scroll-view scroll-x style="white-space:nowrap;height:100%;">
            <view wx:for="{{weather.futureList}}" wx:key="*this" style="width:30%;
                height:100%;display:inline-block;border-right:2rpx solid #edebe9;
                color:#bbb;">
```

```
                <view class="center" style="height: 25%;font-size: 45rpx;">
                    {{item.week}}</view>
                <view class="center" style="height: 40%">
                    <image style="width: 120rpx;height: 120rpx;opacity: 0.5;"
                        src="/static/image/black/{{item.type}}.png"></image>
                </view>
                <view style="text-align: center;height: 10%;">{{item.type}}</view>
                <view class="center" style="height: 25%;font-size: 30rpx;">
                    {{item.wendu}}</view>
            </view>
        </scroll-view>
    </view>
</view>
```

以“weui-”开头的样式，如“weui-flex”，是weui类库定义的通用样式，它们位于sim.js/weui目录之下。除了weui样式之外，pages/index/index页面没有自定义的通用样式，所有样式均以内联的方式直接写在了style属性之上。这样调试改动起来就比较方便，只有在多处或多个页面使用的样式，才需要抽离出来在wxss文件中定义。

在第二个view组件的style属性中，有如下样式：

```
width:750rpx;
```

其中，750rpx代表与屏幕等宽。rpx是responsive pixel的缩写，它是可以根据屏幕大小进行自适应调整的像素单位。小程序规定，屏幕宽度为750。

在小程序开发中，默认使用iPhone 6模拟器，如图4-3所示。iPhone6的屏幕宽度为375px，共有750个物理像素，则750rpx=375px=750物理像素，1rpx=0.5px=1物理像素。

使用iPhone 6模拟器有如下好处：一是分辨率相对较高，可以保证视图在较小的屏幕尺寸上也能保持清晰；二是iPhone 6的屏宽为375px，恰好1px=2rpx，这样在调试样式时容易进行单位换算。

除非有特殊需求，小程序开发中均应使用rpx作为默认的长度单位。

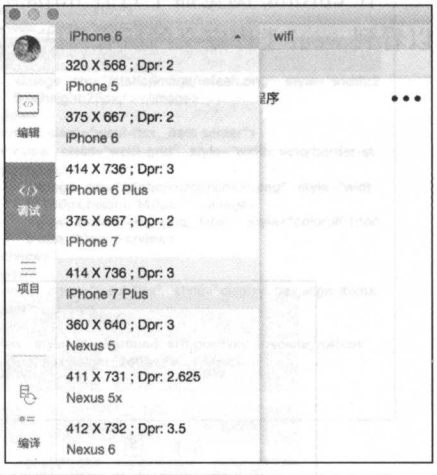


图 4-3

4.1.2 绝对定位

在展示天气状况时，用于实现未来天气列表的容器有一个固定的高度480rpx，而上面其余的UI可随屏幕高度自动伸缩。所以，未来天气列表的容器使用的样式如下：

```
position: absolute;bottom: 0;height: 480rpx;
```

其中，height定义的高度为480rpx，position将定位属性设置为absolute，意即相对于

父容器进行绝对定位。bottom 指定距离父容器底部为 0。

注意 CSS 知识点

position 属性有两个表示绝对定位的值，一为 absolute，在上面已经用到了；一为 static。不同之处在于，前者是相对于父容器的绝对定位，后者是相对于浏览器或屏幕窗口的绝对定位。

第二个 view 容器用于实现随手机屏幕自动伸缩的样式，代码如下：

```
position:absolute;bottom:480rpx;top:0
```

其中，bottom 设置为 480rpx，恰是未来天气列表容器的高度；top 等于 0，代表与屏幕顶端对齐。

这样就实现了绝对定位。

4.2 如何使用 weui

weui-flex 样式是 weui 类库定义的样式，是一个横向满屏的容器，它与子样式 weui-flex__item 结合可以实现横向平均分布的布局，如图 4-4 所示。

在 chrome 浏览器中打开 https://weui.io，然后打开开发者工具，并切换至手机模式，可以看到 weui 类库定义的所有样式，如图 4-5 所示。

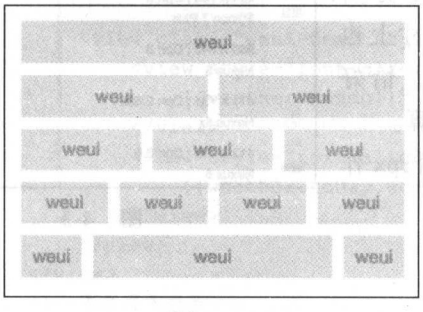


图 4-4

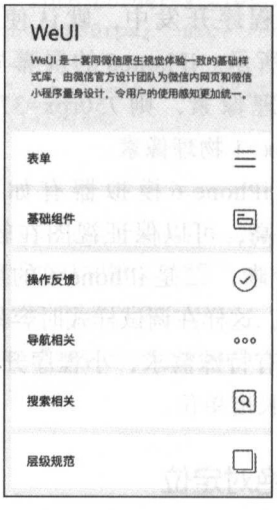


图 4-5

在 sim.js 类库中嵌入的 weui 是基于 Web weui 改写的 wxss weui，项目地址是 https://github.com/weui/weui-wxss。wxss weui 与 Web weui 实现了相同的视觉效果，在浏览器上查看 https://weui.io，类似于在手机上查看“WeUI for 小程序”。

扫描下方二维码，可以体验微信官方的 wxss weui 组件小程序：

官方小程序示例相比于 weui.io 多了媒体组件、地图、画布等功能。

善用 wxml 面板

基于 wxss weui 可以帮助初学者快速建立小程序的 UI，遇到 weui 样式不能满足需求的，使用 style 属性改写样式即可，示例代码如下：

```
<view class="weui-grid" style="width: auto;border-style: none;">
    <image style="width: 320rpx;height: 280rpx;" src="/static/image/white/
        {{weather.today.type}}.png"></image>
    <view class="weui-grid__label" style=
        "color:#fff;font-size: 40rpx;">
        {{weather.today.type}}</view>
</view>
```

这里使用了 weui-grid 实现“上面是一个图像，下方是一个文本并居中对齐”的布局，不过，由于 weui-grid 具有边框，因此这里使用了“ border-style: none”将之去除。

在微信开发者工具中，单击调试工具区左上角的“选择”按钮，使之变成蓝色，如图 4-6 所示。

此时在模拟器上点中任何视图元素，均会在 Wxml 面板中自动选择该元素。直接在 Wxml 面板中修改样式，待效果完善，然后将之复制至代码中即可。

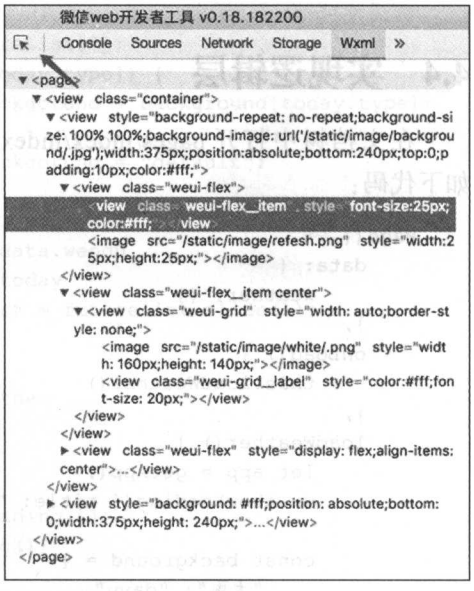


图 4-6

注意 在旧版本的微信开发者工具中，点选视图元素的操作很不友好，如果当前视窗不是 Wxml 面板，工具会提示用户“请先切换至 Wxml 面板”，而不会自动切换。

4.3 关于 static 目录

在 pages/index/index.wxml 文件中，使用了图像组件 image，以“ static/image”开头的 src 属性，使用的是绝对地址。每个小程序项目都相当于是微信网站的子域名网站，如

图 4-7 所示。

```
▶ Failed to load image http://318341983.debug.open.weixin.qq.com/static/image/background/.jpg : the
server responded with a status of 404 (HTTP/1.1 404 Not Found)
From server 127.0.0.1
```

图 4-7

图 4-7 所示的这条 console 面板打印出来的错误消息，是由于没有找到图片资源。但借此可以看出，该项目位于子域名 318341983.debug.open.weixin.qq.com 下。

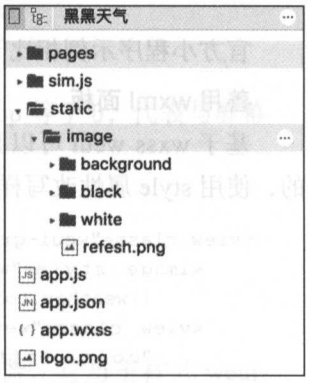


图 4-8

下载项目源码，解压，将源码中的 static 目录复制到小程序项目根目录下。完成后的文档树如图 4-8 所示。

static 目录一般包含 image、style、js 等子目录，分别代表图像、样式、脚本。图标资源一般放在 static/image/icon 下。遵照简单易读易维护的原则，资源目录应使用单词的单数形式。

4.4 实现逻辑层

在文档树中打开 pages/index/index.js 文件，将内容替换为如下代码：

```
Page({
    data: {
        weather: {}
    },
    onLoad() {
        this.loadWeather()
    },
    loadWeather() {
        let app = getApp()
        wx.showLoading({ title: "加载中" })

        const background = {
            "大雨": "dayu",
            "中雨": "dayu",
            "小雨": "xiaoyu",
            "暴雨": "dayu",
            "雷阵雨": "leizhenyu",
            "晴": "qing"
        };
        // 先以ip查询出城市
        let requestCity = app.request('http://int.dpool.sina.com.cn/iplookup/
           iplookup.php?format=json').then(
            res => {
                var weather = {};
```

```
                var city = res.city;
                weather.city = city;
                app.request(`http://wthrcdn.etouch.cn/weather_mini?city=
                    ${city}`).then(
                    res => {
                        if (res.status === 1000) {
                            var data = res.data;
                            var forecast = data.forecast;        //天气特征与未来天气
                            var futureList = [];
                            for (var i = 0; i < forecast.length; i++) {
                                var one = forecast[i];
                                var future = {
                                    week: one.date.slice(-3),
                                    type: one.type,
                                    wendu: one.low.split(' ')[1] + "-" + one.high.
                                        split(' ')[1]
                                }
                                futureList.push(future);
                            }

                            var today = futureList[0];
                            if (background[today.type]) {
                                today.typeBackgorund = background[today.type];
                            } else {
                                today.typeBackgorund = "default";
                            }

                            weather.wendu = data.wendu
                            weather.today = today
                            weather.futureList = futureList.slice(1);

                            this.setData({
                                weather: weather
                            });
                        }
                        wx.hideLoading()
                    }).catch(() => wx.hideLoading())
            }).catch(() => wx.hideLoading())
    }
})
```

上述代码先是调用 http://int.dpool.sina.com.cn/iplookup/iplookup.php?format=json 这个接口，依据所在 ip 地址信息获取所在城市信息。

然后调用如下接口，依据城市查询当地天气数据：

```
'http://wthrcdn.etouch.cn/weather_mini?city=${city}'
```

在上面的代码中，使用（``）符号而非双引号（""）或单引号（''），可以轻松实现变量注入，将变量 city 注入到字符串中。

4.4.1 js 函数 split 与 push

以下代码是将字符串 one.low（例如“低温 24℃”）以空格字符拆分为数组，再获取其第二个元素：

```
one.low.split(' ')[1]
```

下述代码是向数组 futureList 中推入新元素 future：

```
futureList.push(future)
```

这行代码

```
catch(() => wx.hideLoading())
```

等同于：

```
catch(() => {wx.hideLoading()})
```

因为是单行，所以可以将外围的花括号 {} 略去。

在文档树中打开 app.json，将小程序的标题修改为“黑黑天气”。由于代码中使用了 http 接口，无法添加到服务器安全域名，故不能提交审核。

4.4.2 下载源码

在本节的学习过程中，如果遇到问题，可以通过如下方式来解决。

- 加入小程序微信群与作者及其他读者一同探讨。微信扫描前言中的二维码，关注“艺述思维”，发送“小程序”进群。
- 下载作者的源码，对照查找问题。在公众号中发送“黑黑天气 4.4”获取下载地址。

第 5 章 Chapter 3

笑林百家

本章将实现一个带有 tabBar 导航的笑话小程序。如图 5-1 所示，这个小程序共有两个页面，第一个页面是笑谈，第二个页面是趣图。无论是哪个页面，向下滚动到底部时，都可以触发加载更多内容。图 5-2 是单击图片放大预览的效果。

前往 https://mp.weixin.qq.com，按照 1.1 节的方法，注册小程序账号。将名称修改为“笑林百家”，将介绍修改为“笑话、趣图”，在服务类目中选择“工具”→“图片 / 音频 / 视频”。另外，按照 2.9.2 节的方法，制作如图 5-3 所示的头像，并修改之。

打开微信开发者工具，按照 3.1 节的方法，基于 sim.js 项目模板，创建初始项目。

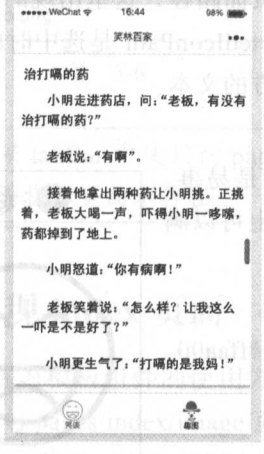


图 5-1

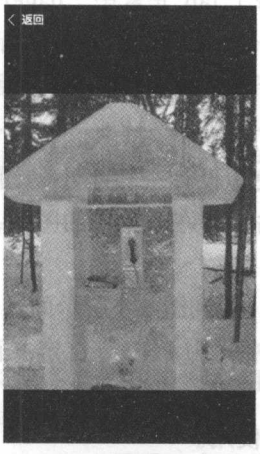


图 5-2

图 5-3

5.1 使用 tabBar

在文档树中打开 app.json，将 navigationBarTitleText 修改为“笑林百家”的方法如下。

在 window 配置下方，添加一个 tabBar 配置，代码如下：

```
"tabBar": {
    "color": "#000000",
    "selectedColor": "#ffaa00",
    "borderStyle": "white",
    "backgroundColor": "#ffffff",
    "list": [
        {
            "pagePath": "pages/index/index",
            "text": "笑谈",
            "iconPath": "/static/image/icon/joke.png",
            "selectedIconPath": "/static/image/icon/joke_on.png"
        },
        {
            "pagePath": "pages/index/image",
            "text": "趣图",
            "iconPath": "/static/image/icon/funny.png",
            "selectedIconPath": "/static/image/icon/funny_on.png"
        }

    ]
}
```

在上述代码中，tabBar 用于设置小程序底部的导航栏。子字段 color 代表 tabBar 默认的文本颜色。selectedColor 是当前 tab 被选中时的文本颜色。borderStyle 是上边框颜色，支持 white 和 black 的颜色，一般设置为白色。backgroundColor 是 tabBar 的背景底色。

list 是一个 tab 的集合，在每个 tab 元素中，pagePath 是页面路径，这里出现的 page 在 app.json 中必须要有定义。iconPath 是默认的常态图标，selectedIconPath 是选中时的图标。tabBar 中使用的图标大小应不低于 81px 见方。text 是图标下方的文本。

使用 icon

如果不是设计师，那么即使是设计一枚小小的图标也不是易事。http://www.flaticon.com/ 这个网站上有许多免费的图标，基本上可以满足所有初学者的开发需求。

在 flaticon 网站上搜索 joke，找到图 5-4 所示的这个图标。

然后在下方依次选择文件格式为 PNG，自定义颜色为 #ffaa00，大小为 128，如图 5-5 所示。

最后，将其保存至本地即可使用。

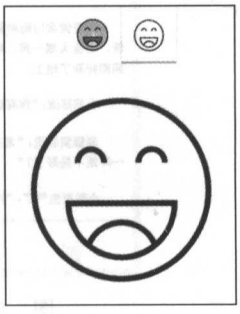


图 5-4

此时，app.json 中定义 tabBar 所需的 4 张图标还不存在。为了简单起见，读者可使用作者源码中的图片。从本章的 5.3.4 节下载最终

源码，解压，将 static 目录复制到自己的项目根目录下即可使用。完成后的文档树如图 5-6 所示。

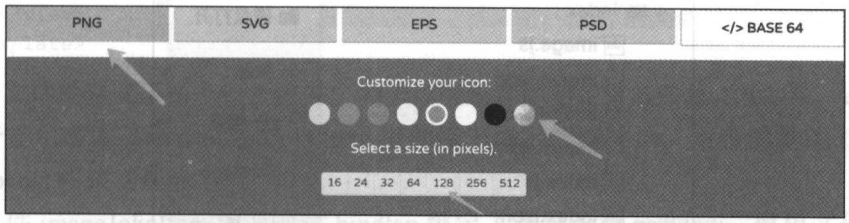


图 5-5

缺失的不仅是图标文件，还有 page。在 tabBar 中定义的 page，必须要存在于 app.json 的 pages 集合中，使用 2.1.3 节介绍的快捷创建页面的方法，创建 pages/index/image，作为趣图页面。这一步完成之后，效果如图 5-7 所示。

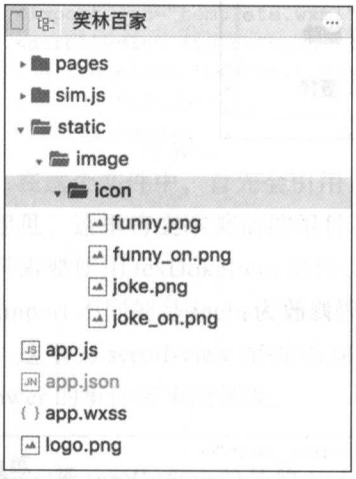


图 5-6

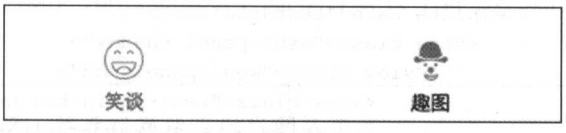


图 5-7

注意 如果 tabBar 中使用的 page 在 pages 配置中缺失，那么 tabBar 仍可以显示，但小程序会提示页面不存在。

5.2 实现 index 页面

为了练习模板组件的使用方法，本节将定义一个模板组件，同时提供给 pages/index/index 页面与 pages/index/image 页面使用。

如图 5-8 所示，在文档树中找到 pages/index 目录，在此目录下新增一个 template.wxml 单文件，笔者随后将在这个文件中创建模板组件。

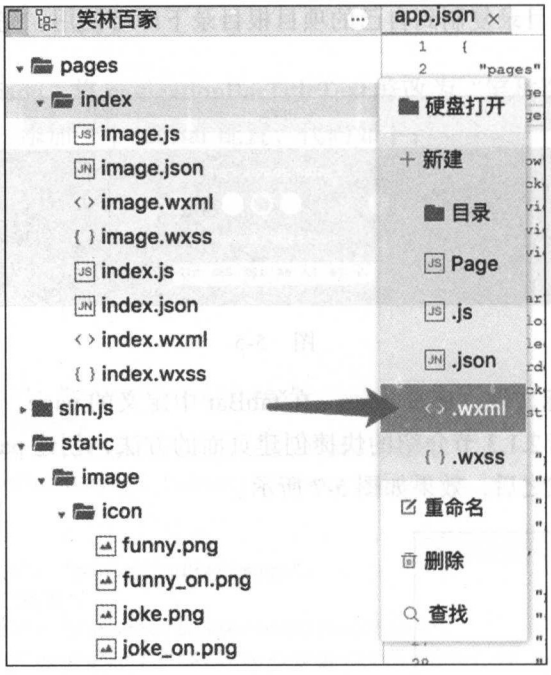


图 5-8

5.2.1 定义模板组件

在定义模板组件时，将 template.wxml 文件的标签代码修改为：

```
<template name="textJokeItem">
    <view class="weui-panel shadow">
        <view class="weui-panel__bd">
            <view class="weui-media-box weui-media-box_text">
                <view class="weui-media-box__title weui-media-box__title_in-
                    text">{{item.title}}</view>
                <text>{{item.content}}</text>
            </view>
        </view>
    </view>
</template>

<template name="imageJokeItem">
    <view class="weui-panel">
        <view class="weui-panel__bd">
            <view class="weui-media-box weui-media-box_text">
                <view class="weui-media-box__title weui-media-box__title_in-
                    text">{{item.title}}</view>
                <view class="weui-media-box__desc">
                    <image src="{{item.sourceurl}}" mode="aspectFit" bindtap=
                        "preview" data-url="{{item.sourceurl}}"/>
```

```
                </view>
            </view>
        </view>
    </view>
</template>
```

这个文件中定义了两个模板组件。关于模板组件的相关讲解参见 2.6.2 节。textJokeItem 组件用于显示文本笑话，imageJokeItem 组件用于展示趣图笑话。两者都是基于 weui 类库的 weui-panel 样式实现的。

在组件 imageJokeItem 中，通过 bindtap 绑定了预览函数 preview。模板文件不存在逻辑层 js 文件，该函数必须由组件的调用方传递进来。

5.2.2 import 与 include 的区别

在文档树中打开 pages/index/index.wxml 文件，将标签代码替换为：

```
<import src="template.wxml"/>
<scroll-view style="height:100%" scroll-y bindscrolltolower="loadMore">
    <template is="textJokeItem" data="{{item}}" wx:for="{{jokeList}}" wx:key=
        "title"/>
</scroll-view>
```

在这个文件中，首先会引用 import 标签，然后会引入模板文件 template.wxml。为了简单起见，这里将文本笑话的组件与图片笑话的组件定义在了一个文件中。虽然在这个页面中只需要使用 textJokeItem 组件，但并不会造成冗余和浪费。import 是“引用”式的引用，与 import 不同的是 include 导入，后者是将目标文件的代码复制到 include 标签所在的位置。

这里为 scroll-view 组件添加了 height 等于 100% 的样式，如果没有这个样式，scroll-tolower 的事件将不能触发。

练习 将 scroll-view 组件的 style 属性去除，然后观察能否触发触底事件。

5.2.3 js 数组函数

在文档树中打开 pages/index/index.js 文件，将代码替换为：

```
Page({
    data: {
        page: 1,
        jokeList: [],
    },
    onLoad(options) {
        this.retrieve()
    },
    retrieve() {
        let app = getApp()
```

```
        var url = `http://api.laifudao.com/open/xiaohua.json?page=${this.data.page}
           &size=10`
        app.request(url).then(
            res => {
                console.log(res)
                res = res.map(item => ({title:item.title,content:item.content.
                  replace(/<br\/>/g,'\n')}) )
                this.setData({
                    jokeList: this.data.jokeList.concat(res)
                })
            })
    },
    loadMore() {
        ++this.data.page
        this.retrieve()
    }
})
```

在页面初始数据 data 中，page 表示当前页码，jokeList 用于储存加载到的笑话列表，作为视图层渲染之用。

注意 由于本章小程序所用的笑话 API 并不支持分页查询，所以关于 page 的使用仅是模拟正常调用。读者在调试代码的过程中会发现，后续加载的内容与前面的相重合。

下面针对上述代码进一步说明。

1. js 数组函数 concat

下面这行代码使用了 js 的 concat 函数。concat 意即合并，它会将两个或多个数组合并为一个数组并返回：

```
this.data.jokeList.concat(res)
```

打开接口的网址 http://api.laifudao.com/open/xiaohua.jso，可以看到返回的数据库包含了“
”，这是 html 换行标签。在小程序中，尚不存在可以正常解析它的组件。解决方法唯有将它转换为换行符“\n”，小程序的 text 组件支持换行符“\n”的渲染。

2. js 数组函数 map

在下面这行代码中，至少包括了三个值得一提的知识点：

```
res.map(item => ({title:item.title,content:item.content.replace(/<br\/>/g,'\n')}) )
```

1）map 是 js 自带的函数，它将目标数组的元素依次传递给指定的函数，经函数处理后组成新的数组并返回。

2）以下代码是一个简写的箭头函数，形参是 item。

```
item => (..)
```

3）当箭头函数的代码仅有一行时，return 关键字可以省略，以括号 () 表示返回。

5.2.4 js 正则表达式

以下代码将 content 中的“
”替换为“ \n”。replace 是 js 自带的替换方法，第一个参数既可以是字符串文本，也可以是正则表达式，在这里笔者使用的是正则表达式。g 代表全局，即替换所有匹配项。在 js 正则表达式中，“ /”属于特殊字符，在使用时须用“ \”转义为字面量。

```
item.content.replace(/<br\/>/g,'\n')
```

正则表达式是用于匹配字符串中字符组合的模式，其由包含在斜杠之间的模式组成，如下所示：

```
const regex = /ab+c/;
const regex = /^[a-zA-Z]+[0-9]*\W?_$/gi;
```

特殊字符在正则表达式中具有特殊的规定意义，并非字符本身的字面量意义。下面是 js 正则表达式中的特殊字符列表。

- \：在非特殊字符之前的反斜杠表示下一个字符是特殊的，不能从字面上解释。或者将其后的特殊字符，转义为字面量。
- ^：匹配输入的开始。 如果多行标志被设置为 true，那么也要匹配换行符后紧跟的位置。
- $：匹配输入的结束。如果多行标志被设置为 true，那么也要匹配换行符前的位置。
- *：匹配前一个表达式 0 次或多次。等价于 {0,}。
- +：匹配前面一个表达式 1 次或多次。等价于 {1,}。
- ?：匹配前面一个表达式 0 次或 1 次。等价于 {0,1}。 如果紧跟在任何量词“ *”“ +”“ ?”或“ {}”的后面，将会使量词变为非贪婪的（匹配尽量少的字符），这点与默认使用的贪婪模式（匹配尽可能多的字符）正好相反。 例如：“123abc”应用 /\d+/ 将会返回“123”，如果使用 /\d+?/，那么就只会匹配到“1”。
- .：（小数点）匹配除换行符之外的任何单个字符。
- x | y：匹配‘x’或‘y’。
- {n}：n 是一个正整数，匹配前面一个字符刚好发生了 n 次。
- {n,m}：n 和 m 都是正整数。匹配前面的字符至少 n 次，最多 m 次。如果 n 或 m 的值是 0，那么这个值将被忽略。
- [xyz]：一个字符集合。匹配方括号中的任意字符，包括转义序列。可以使用破折号（-）来指定一个字符范围。对于点号（.）和星号（*）这样的特殊符号在一个字符集中没有特殊的意义。他们不必进行转义，不过转义也是起作用的。
- [^xyz]：一个反向字符集。也就是说，它匹配任意一个没有包含在方括号中的字符。

你可以使用“-”来指定一个字符范围。任何普通字符在这里都是起作用的。

- \d：匹配一个数字。等价于 [0-9]。
- \D：匹配一个非数字字符。
- \s：匹配一个空白字符，包括空格、制表符、换页符和换行符。
- \S：匹配一个非空白字符。
- \w：匹配一个单字字符（字母、数字或者下划线）。等价于 [A-Za-z0-9_]。
- \W：匹配一个非单字字符。等价于 [^A-Za-z0-9_]。

5.3 实现 image 页面

在文档树中打开 pages/index/image.wxml 页面，将标签代码替换为：

```
<import src="template.wxml"/>

<scroll-view style="height:100%" scroll-y bindscrolltolower="loadMore">
    <template is="imageJokeItem" data="{{item,preview}}" wx:for="{{picList}}"
        wx:key="title"/>
</scroll-view>
```

这个页面引入了模板文件 template.wxml，使用预定义的 imageJokeItem 模板，循环渲染了 picList 数组，完成了趣图列表的展示。

5.3.1 将函数作为参数传递

pages/index/image.wxml 页面的标签代码与 pages/index/index.wxml 中的代码类似。在模板调用代码中，通过 data 不仅传递了数据对象 item，还传递了一个页面函数 preview。在 wxml 页面中，可以在事件绑定中使用函数，在模板组件的数据绑定中，也可以使用函数。不同之处在于，事件绑定的函数不需要放在花括号（{}）中，而此处的 preview 须置于花括号之内。

在 js 中，函数 Function 可以与字符串 String、对象 Object 等视为同等地位的数据，也可以在函数中作为参数传递，或者作为结果返回。

5.3.2 关于 lower-threshold 属性

scroll-view 是可滚动视图容器组件，scroll-y 代表竖向滚动，lower-threshold 代表距离底部多远时，触发 scrolltolower 事件。如果设置为 100，则代表距离底部不足 100 时触发该事件。默认为 50，代表距离底部不足 50 时触发该事件。

这里将设置 lower-threshold 为 100，lower-threshold 的单位是 px。

在 app.js 中设置的 tabBar，占据了屏幕 100rpx 的高度。这使得每个页面的屏幕高度减少了 100rpx，但 lower-threshold 是从 tabBar 的上边缘开始计算的，所以有没有 tabBar 并不会对 scroll-view 在触发 scrolltolower 事件上造成影响。

loadMore 将是我们在逻辑层定义的函数，用于加载更多内容。

5.3.3 使用 wx.previewImage 接口

在文档树中打开 pages/image/image.js 文件，将其内容替换为如下代码：

```
Page({
    data: {
        page: 1,
        picList: []
    },
    retrieve() {
        let app = getApp()

            var url = `http://api.laifudao.com/open/tupian.json?page=${this.data.
               page}&size=10`
        app.request(url).then(
            res => {
                this.setData({
                    picList: this.data.picList.concat(res)
                })
            })
        },
        onLoad(options) {
            this.retrieve()
        },
        // 滑动到底部加载更多
        loadMore() {
            ++this.data.page
            this.retrieve()
        },
        preview(e) {
            var urls = [e.target.dataset.url]
            wx.previewImage({
                urls: urls
        })
    }
})
```

函数 preview 使用了微信小程序的图像预览接口 wx.previewImage。图像预览体验与微信中的类似。

由于使用了 http 接口，所以这个项目不能提交审核。

5.4 下载源码

在本节的学习过程中，如果遇到问题，可以通过如下方式来解决。

- 加入小程序微信群与作者及其他读者一同探讨。微信扫描前言中的二维码，关注“艺述思维”，发送“小程序”进群。
- 下载作者的源码，对照查找问题。在公众号中发送“笑林百家 5.3”获取下载地址。

Chapter 6

第 6 章

图灵聊聊

本章主要实现一个联系人列表，并基于图灵 API 实现自动聊天功能，如图 6-1 和图 6-2 所示。

在该聊天小程序里，首页实现了下拉更新，向下滚动可以加载更多的功能。

首先，前往 https://mp.weixin.qq.com，按照 1.1 节的方法，注册小程序账号。将名称修改为“图灵聊聊”，将介绍修改为“自动聊天小工具”。然后按照 2.9.2 节介绍的方法，制作图 6-3 所示的头像，并修改之。

然后，打开微信开发者工具，按照 3.1 节介绍的方法，基于 sim.js 项目模板创建初始项目。

接着，在文档树中打开 app.json，将小程序标题修改为“图灵聊聊”。

图 6-1

图 6-2

图 6-3

从本章 6.6 节下载源码，解压，获取 static 资源目录，并复制到“图灵聊聊”项目的根目录下。其中，static/image/icon 目录下的图标文件，可以参考本书 5.1 节在“使用 icon”中提到的方法快速生成。

在文档树中打开 app.json，参照 5.2 节的方法设置 tabBar，代码如下：

```
"tabBar": {
    "color": "#929292",
    "selectedColor": "#ff9630",
    "backgroundColor": "#f7f7f8",
    "borderStyle": "white",
    "list": [{
        "pagePath": "pages/index/index",
        "text": "首页",
        "iconPath": "/static/image/icon/home-outline.png",
        "selectedIconPath": "/static/image/icon/home-selected.png"
    }, {
        "pagePath": "pages/index/contact",
        "text": "朋友",
        "iconPath": "/static/image/icon/contacts-outline.png",
        "selectedIconPath": "/static/image/icon/contacts-selected.png"
    }, {
        "pagePath": "pages/index/my",
        "text": "我的",
        "iconPath": "/static/image/icon/my-outline.png",
        "selectedIconPath": "/static/image/icon/my-selected.png"
    }]
}
```

在以上代码中，定义了“首页”“朋友”“我的”三个 tab，完成后效果如图 6-4 所示。

图 6-4

6.1 实现 index 页面

在文档树中打开 pages/index/index.wxml 文件，将标签代码替换为如下内容：

```
<scroll-view style="height:100%" scroll-y bindscrolltolower="scrollToLower">
    <view wx:for="{{timeline}}" wx:key="created_at" class="weui-panel weui-panel_
        access">
        <view class="weui-panel__hd no-padding">
            <view class="weui-media-box weui-media-box_appmsg">
                <view class="weui-media-box__hd weui-media-box__hd_in-appmsg">
                    <image class="weui-media-box__thumb" src="{{item.avatar}}" />
                </view>
                <view class="weui-media-box__bd weui-media-box__bd_in-appmsg">
                    <view class="weui-media-box__title">{{item.nickname}}</view>
                    <view class="weui-media-box__desc">{{item.time}}</view>
                </view>
            </view>
```

```
            </view>
            <view class="weui-panel__bd">
                <view class="weui-media-box weui-media-box_text no-padding-top">
                    <view class="weui-media-box__title weui-media-box__title_in-
                        text"></view>
                    <view class="weui-media-box__desc">{{item.text}}</view>
                    <image style="width:100%;margin-top:20rpx;" data-original-pic="{{item.
                        original_pic}}" wx:if="{{item.original_pic}}" src="{{item.
                        original_pic}}" mode="aspectFill" catchtap="previewImage">
                        </image>
                     <view class="weui-media-box__info">
                        <view class="weui-media-box__info__meta">喜欢</view>
                        <view class="weui-media-box__info__meta weui-media-box__
                            info__meta_extra">评论</view>
                        <view class="weui-media-box__info__meta weui-media-box__
                            info__meta_extra">转发</view>
                    </view>
                </view>
            </view>
        </view>
    </scroll-view>
```

在这里我们使用了 scroll-view 组件，并通过 style 设置了它的高度为 100%。使用 wx:for 循环渲染数组 timeline，关于 wx:for 标签的详细讲解请参考 2.4.1 节。

之后，将 scrolltolower 事件绑定到 scrollToLower 函数，该函数将在逻辑层定义，用于触发下一页数据的拉取。

接着，使用 catchtap 在 image 组件上将 tap 事件绑定到 previewImage 函数，该函数将调用微信小程序的图片预览接口。使用 bind 绑定的事件是冒泡的，使用 catch 绑定的事件是不冒泡的，详情请见本书的 3.3.1 节。

最后，wx:key 将列表渲染的主键字段定义为“created_at”。在列表渲染中，一般以“id”等字段作为主键。但在本章的示例中，数据是伪造的且存在重复加载，所以在这里以“created_at”作为渲染主键。

6.1.1 建立 server 目录模拟服务器数据

从本章的 6.6 节所附的源码中，将 server 目录复制到项目的根目录下，完成后如图 6-5 所示。

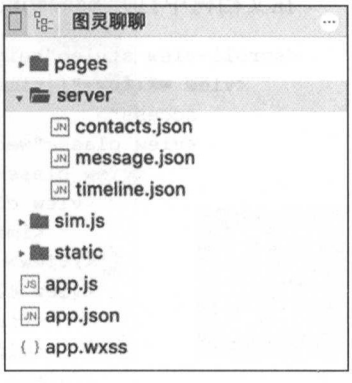


图 6-5

server 目录放置的是模拟数据，在没有服务器端的情况下，可便于进行小程序前端测试。

在文档树中打开 pages/index/index.js 文件，将内容替换为如下代码：

```
let app = getApp()

Page({
```

```
    data: {
        timeline: [],
        page:1
    },
    onLoad() {
        this.retrieve()
    },
    onPullDownRefresh() {
        this.data.page = 1
        this.data.timeline.length = 0
        this.retrieve().finally(_ => wx.stopPullDownRefresh())
    },
    scrollToLower() {
        this.data.page++
        this.retrieve()
    },
    retrieve() {
        return app.request('http:// 1482862701.debug.open.weixin.qq.com/server/
            timeline.json?page=${this.data.page}').then(res => {
            let timeline = this.formatTimeline(res.data)
            this.setData({
                timeline: [...this.data.timeline, ...timeline]
            })
        })
    },
    formatTimeline(items) {
        let now = new Date().getTime()
        var i = 0
        items.forEach(item => {
            item.avatar = `/static/image/${item.avatar % 4}.jpeg`
            item.created_at = now - (i++) * 6 * 60 * 60 * 1000
            item.time = app.humanFormatTime(item.created_at)
        })
        return items
    },
    previewImage(event) {
        wx.previewImage({
            current: '',
            urls: [event.target.dataset.originalPic]
        })
    }
})
```

其中，函数 retrieve 用于加载时间线数据，这里临时用到了一个域名，如下所示：

```
1482862701.debug.open.weixin.qq.com
```

这个 5 级域名是本章小程序示例的调试域名。每个小程序都是微信服务器上的一个特殊的 Web 站点，都有一个调试域名。下面在视图层的任何一个 image 组件上，以 src 属性编写一个不存在的图片地址，如下所示：

```
<image src="/static/xx.jpg" />
```

Console 面板便会报错，如图 6-6 所示。

```
▶ Failed to load image http://1482862701.debug.open.weixin.qq.com/static/xx.jpg : the server
responded with a status of 404 (HTTP/1.1 404 Not Found)
From server 127.0.0.1
```

图 6-6

图 6-6 所示的出错地址里就包含了所要找的调试域名。

注意 一个小程序并不只有一个固定的调试地址。调试域名可以帮助用户在没有服务器的情况下，加载本地的文件，这样会更方便，但仅适合在开发时使用。

6.1.2 在文件作用域中声明 app

由于在多处用到了 app，所以将如下这行代码写在了页面顶部、Page 外面：

```
let app = getApp()
```

在 js 文件中，声明的变量和函数只在该文件中有效，不同的 js 文件可以声明不同的变量和函数，并不会相互影响，而这是因为文件作用域起了作用。

在这里，页面声明生成的 app 变量，在本 js 文件中有效，在 Page 对象的任何函数中均可以引用，但是在其他页面的 js 文件中如有需要必须另外声明。

6.1.3 调用图像预览接口

在 previewImage 函数中，调用了微信小程序的图片预览接口 wx.previewImage。在该接口接受的参数对象中，current 表示默认显示的图片，urls 表示预览的图片列表。urls 不能为空，但 current 可以为空，甚至可以不传，如下所示：

```
previewImage(event) {
    wx.previewImage({
        urls: [event.target.dataset.originalPic]
    })
}
```

6.1.4 用户友好的时间格式化方法 formatTimeline

函数 formatTimeline 有两个作用：一是格式化时间，生成类似于“1 小时前”“刚刚”“几天前”这样的格式；二是组合用户的头像地址。

在函数 formatTimeline 中，下面这行代码的意思是以 4 整除 item.avatar，余几。

```
item.avatar % 4
```

其中，“%”在 js 中是取模运算符。下面这行代码旨在拼出一个用户头像的本地图片地

址，是给视图层的 image 组件的 src 属性使用的：

```
item.avatar = `/static/image/${item.avatar % 4}.jpeg`
```

humanFormatTime 是在 sim.js 中定义的一个工具方法，输入毫秒时间，即可返回人类可读易理解的时间。

new Date() 返回当前时间，getTime() 返回时间的毫秒。在 formatTimeline 函数中，笔者人为改变 created_at，是为了让读者看到 humanFormatTime 函数的效果；同时在视图层的 for 循环列表渲染中，避免主键 created_at 重复。

6.1.5 js 语言中的展开符

在函数 retrieve 中，下面这行代码：

```
[...this.data.timeline, ...timeline]
```

使用了 js 语言中的展开符号，“...timeline”意即将数组 timeline 展开，这行代码整体的意思是将旧 timeline 数组与新 timeline 数组先后展开，并将它们组合为新的数组。

使用 2.6.1 节用过的 js 方法 concat，将新旧 timeline 数据连接在一起。

6.1.6 变量自增

函数 scrollToLower 将在 scroll-view 视图滚动到列表底部的时候触发，下面这行代码是使 page 变量自增 1：

```
this.data.page++
```

在正式应用中，分页加载数据需要指定页码。在本章的示例中，this.data.page 仅是一个示例变量，在代码中并无实际作用。

在函数 onPullDownRefresh 中，下面这行代码用于将 timeline 数据清空：

```
this.data.timeline.length = 0
```

6.1.7 js 的忽略符

“_”在 js 中是忽略符，代表“没有”“无定义”和“无返回值”。在 swift 与 golang 语言中，也存在同样的符号及含义。

在 js 中，函数可以用箭头语法“=>”进行定义，这种语法的主要意图是定义轻量级的内联回调函数，书写简便，代码清晰。例如，在下面的代码中，传入 map(…) 中的参数便是一个箭头函数：

```
[5, 8, 9].map((item) => item + 1)
```

当箭头函数有一个参数时，参数两边的括号是可有可无的，所以上面的代码也可以写作：

```
[5, 8, 9].map(item => item + 1)
```

如果箭头函数没有参数，则必须使用圆括号“( )”或者使用忽略符“_”，如下面的代码所示：

```
[5, 8, 9].map( () => { console.log('some code') } )
```

或：

```
[5, 8, 9].map( _ => { console.log('some code') } )
```

使用忽略符，可使箭头函数的书写更加简单方便。

6.1.8 通用的下拉区域

函数 onPullDownRefresh 是微信小程序官方预定义的函数，当用户在页面顶端下拉屏幕时，会自动执行该函数。该函数能起作用的另一个必要条件是在页面的 json 配置文件中开启 enablePullDownRefresh。

在文档树中打开 pages/index/index.json，添加 enablePullDownRefresh 字段，如下所示：

```
{
    "enablePullDownRefresh": true
}
```

若此时刷新项目，然后在小程序顶部下拉屏幕，通常会是无反应的。在微信开发者工具中，同样可以测试 pull down 效果，并非只有在真机上才可以测试。

失效的原因在于没有 pull down 区域，整个屏幕都是 scroll-view 的区域，scroll-view 本身有上下滑动的功能，所以 pull down 事件不能触发。在文档树中打开 pages/index/index.wxml，在底部添加如下标签代码：

```
<view class="weui-flex"
    style="position:absolute;top:0;width:100%;
        height:80rpx;">
    <!--页面顶部通用隐形小刷条-->
</view>
```

这个透明的 pull down 区域是通用的。刷新项目，就能得到相应的效果了，如图 6-7 所示。

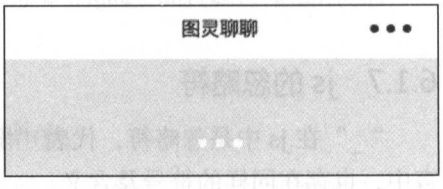


图 6-7

6.2 实现联系人页面

在实现联系人页面时，首先要使用 2.1.3 节快捷创建页面的方法，创建 pages/index/contact 页面。

然后在文档树中打开 pages/index/contact.wxml 文件，将内容替换为如下标签代码：

```
<block wx:for="{{contacts}}" wx:key="nickname">
    <view class="weui-cells__title">{{item.header}}</view>
    <view class="weui-cells weui-cells_after-title">
        <navigator url="message?name={{item.nickname}}" wx:for="{{item.list}}"
            wx:key="nickname" class="weui-cell weui-cell_access">
            <view class="weui-cell__hd ">
                <image src="{{item.avatar}}" style="margin-right: 20rpx;vertical-
                    align: middle;width:80rpx; height: 80rpx; "></image>
            </view>
            <view class="weui-cell__bd ">{{item.nickname}}</view>
            <view class="weui-cell__ft  weui-cell__ft_in-access">{{item.location}}
            </view>
        </navigator>
    </view>
</block>
```

在上述代码中，contacts 是在逻辑层定义的联系群组，header 是分组字母，list 是该分组下的元素。

这里还使用了嵌套的 wx:for，内嵌循环的对象是 item.list。item.avatar 中的 item 与 item.list 中的 item 并非指同一个 item。

相对路径下的页面 message 是自动聊天的窗口，将在下面的代码中进行定义。

在文档树中打开 pages/index/contact.js 文件，将内容替换为如下 js 代码：

```
Page({
    data: {
        contacts: []
    },
    onLoad() {
        let app = getApp()
        app.request('http://1265839903.debug.open.weixin.qq.com/server/contacts.
            json').then(res => {
            this.setData({
                contacts: this.formatContacts(res.data)
            })
        })
    },
    formatContacts(items) {
    var result = []
    var group = { header: '', list: [] }
    for (var i in items) {
        var item = items[i]
        if (item.header) {
            if (item.header != group.header) {
                if (group.list.length > 0) result.push(group)
                group = { header: item.header, list: [] }
            }
        }
```

```
            item.avatar = `/static/image/${item.avatar % 4}.jpeg`
            group.list.push(item)
            }
            return result
        }
})
```

细心的读者不难发现，文件 pages/index/contact.js 中使用的调试地址与 pages/index/index.js 中使用的调试地址并不相同，这说明每个小程序都会有不同的调试地址，调试地址也仅能临时用于开发中。

从服务器取回的数据未必适合直接渲染，在函数 formatContacts 中，会将数据按 header 字段进行分组，具有相同 header 的元素将放在同一个子组中，然后在视图层中使用嵌套的 wx:for 进行渲染。

如下这行代码，用于循环数组 items 中的每个元素：

```
for (var i in items)
```

它和下面的这行代码是等价的：

```
for (var i=0;i<items.length;i++)
```

这是 js 语言中遍历数组的两种方法，相比之下前者更简洁。

6.2.1 js 中的引用传递

如果将如下这行代码：

```
group = { header: item.header, list: [] }
```

改成：

```
group.header = item.header
group.list.length = 0
```

刷新项目，代码不能正常工作。第二种方法存在问题。

从表面上看，第二种方法既赋值了 header 字段，又清空了 list 子数组，效果应当与第一种方法相同。但在 js 语言中，对象（Object）与数组（Array）均是引用传递，第二种方法永远都只是改变同一个 group 对象，并没有生成新的对象，所以所有分组的数据在视图层渲染出来之后均相同。

6.2.2 js 数组的 push 方法

在函数 formatContacts 中，存在如下这行代码：

```
group.list.push(item)
```

其用于将子元素 item 推入数组 group.list 中。push 方法是 js 编程中最常用的操作数组的方法之一，接下来将对它进行详细介绍。

（1）定义

push() 方法可向数组的末尾添加一个或多个元素，并返回新的长度。

（2）语法

```
arrayObject.push(newelement1,newelement2,….,newelementX)
```

第一个参数 newelement1 是必需的，代表要添加到数组的第一个元素。第二个、第三个元素是可选的，可以同时添加多个元素。

push 方法可将它的参数顺序添加到数组的尾部。该方法直接修改原数组，而不是创建一个新的数组。push 方法和 pop 方法使用数组提供的先进后出栈的功能。

（3）示例

在下面的代码中，首先使用 new 关键字创建一个容量为 3 的新数组 arr，然后通过下标访问的方式分别为三个数组元素赋值，接着调用 push 方法向数组尾部推入新元素，此时的容量增加了 1，但数组的内存地址并没有改变。

```
var arr = new Array(3)
arr[0] = "George"
arr[1] = "John"
arr[2] = "Thomas"
arr.push("James")
console.log(arr)
```

以上代码的输出结果为：

```
["George", "John", "Thomas", "James"]
```

6.2.3 接口返回数据的通用格式

在周期函数 onLoad 中，使用 app.request 从本地 server 目录中加载 contacts.json 文件，该文件的内容如下所示：

```
{
    "err_code": 0,
    "err_msg": "success",
    "data": [
        {"nickname":"阿北","location":"北京","avatar":"1","header":"A"},
        {"nickname":"阿木","location":"北京","avatar":"5","header":"A"},
        {"nickname":"阿静","location":"江宁","avatar":"3","header":"A"},
        {"nickname":"阿泰","location":"江宁","avatar":"6","header":"K"},
        {"nickname":"阿静","location":"北京","avatar":"10","header":"K"},
        {"nickname":"阿哲","location":"北京","avatar":"1","header":"M"},
        {"nickname":"阿星","location":"江宁","avatar":"4","header":"O"},
        {"nickname":"阿成","location":"云南","avatar":"5","header":"O"}
    ]
}
```

header 字段的值并不是 nickname 字段的首字母，这里仅是为了测试分组的功能而随意

指定的。

服务器返回的数据，一般包括三个部分：错误码、错误消息、数据。对应于 contacts.json 文件中的字段分别是：err_code、err_msg 和 data。

6.3 实现聊天页面

使用本书 2.1.3 节快捷创建页面的方法，创建 pages/index/message 页面。

在文档树中打开 pages/index/message.wxml 文件，将内容替换为如下标签代码：

```
<view style="padding:0 30rpx;padding-bottom:100rpx;">
    <block wx:for="{{messages}}">
        <view class="message flex-row {{item.from === 'sent' ? 'message-sent' :
            ''}}">
            <view class="message-text {{item.from === 'sent' ? 'message-sent-
                text' : 'message-received-text'}}">
                <text wx:if="{{!item.image}}">{{item.text}}</text>
                <image wx:else mode="aspectFit" catchtap="previewImage" data-
                    image="{{item.image}}" src="{{item.image}}" class="message-
                    image"></image>
                </view>
            </view>
        </block>
    </view>
    <view class="weui-flex" style="box-sizing: border-box;width:100%;position:fixed;
        bottom:0;padding:10rpx;height:80rpx;background-color:#e7e7e7;">
        <view class="weui-flex__item">
            <input class="message-input" value="{{inputContent}}" bindchange=
                "bindChange" />
        </view>
        <button style="margin-top:0;margin-left:10rpx;padding:0 20rpx;" bindtap=
            "sendMessage" class="weui-btn mini-btn" type="primary" size= "mini">
            发送</button>
</view>
```

message 页面的视图分为上、下两部分，上面是聊天消息列表，如图 6-8 所示。

下面是一个文本输入框，固定在屏幕底端，如图 6-9 所示。

图 6-8

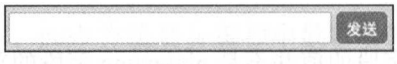


图 6-9

6.3.1 在视图渲染中使用三目运算符

下面这行代码使用了 js 语言的三目运算符：

```
{{item.from === 'sent' ? 'message-sent' : ''}}
```

在这里，如果 item.from 等于 sent，则使用 message-sent 样式；如果不等于，则返回空。

在小程序视图层的渲染代码里，如果想实现条件渲染，除了使用 wx:if 标签之外，另一个方法便是使用三目运算符，使用三目运算符可以减少标签代码量，使代码更加简洁清晰。

6.3.2 js 中的全等于与等于运算符

在 js 中，=== 是全等于运算符，要求类型与值均相同；== 是等于运算符，即使类型不同，只要值相等，也返回 true。例如，下面这行代码将返回 true：

```
"123" == 123
```

在下面这行代码中，item.from 必须全等于‘sent’，字符串 message-sent 才会被渲染。

```
{{item.from === 'sent' ? 'message-sent' : ''}}
```

在开发中，如果能够确定运算符右值的类型，则建议使用全等于运算符；反之，则使用等于运算符。后者在编程中具有更大的灵活性，但也可能埋下难以察觉的 bug。目前流行的高级编程语言，如 Go 语言、Swift 语言、Java 语言等，无一不是强类型语言。强类型，即每个变量的类型在运行时都是确定的。js 不是强类型语言，但在编程规范和习惯上仍可以向强类型语言看齐。

6.3.3 wx:if 条件渲染

下面的这两行代码使用了条件渲染：

```
<text wx:if="{{!item.image}}">{{item.text}}</text>
<image wx:else mode="aspectFit" catchtap="previewImage" data-image="{{item.
    image}}" src="{{item.image}}" class="message-image"></image>
```

这里的 wx:if 所表示的意思是：如果 item.image 有值，不为空，则渲染 text 组件；否则，执行 wx:else，渲染 image 组件。

value="{{inputContent}}" 绑定的是文本框的输入，inputContent 是定义在 data 上的变量之一。bindchange="bindChange" 是将值变化事件绑定到函数 bindChange 上。

6.3.4 使用 css 遮罩实现消息框样式

在文档树中打开 pages/index/message.wxss 文件，将内容替换为如下样式代码：

```
.message{
    margin: 28rpx 0;
    box-sizing: border-box;
```

```
}
.message-text{
    font-size: 32rpx;
    border-radius: 40rpx;
    display: inline-block;
    background-color: #e5e5ea;
    padding: 20rpx 30rpx;
    max-width: 70%;
    word-break: break-all;
    border-radius: 1pc 1pc 1pc 0;
    -webkit-mask-box-image: url("data:image/svg+xml;charset=utf-8,<svg height='35'
        viewBox='0 0 96 70' width='48' xmlns='http://www.w3.org/2000/svg'><path
        d='m96 35c1 7-5 37-42 35-37 2-43-28-42-35-1-7 5-37 42-35 37-2 43 28 42
        35z'/><path d='m0 70c6-2 12-10 12-19v-16l14 27s-8 8-26 8z'/></svg>") 50%
        42% 46% 56%;
}
.message-sent-text{
    padding-right: 40rpx;
    background-color: #00d449;
    border-radius: 1pc 1pc 0 1pc;
    -webkit-mask-box-image: url("data:image/svg+xml;charset=utf-8,<svg height='35'
        viewBox='0 0 96 70' width='48' xmlns='http://www.w3.org/2000/svg'><path
        d='m84 35c1 7-5 37-42 35-37 2-43-28-42-35-1-7 5-37 42-35 37-2 43 28 42
        35z'/><path d='m96 70c-6-2-12-10-12-19v-16l-14 27s8 8 26 8z'/></svg>") 50%
        56% 46% 42%;
}
.message-received-text{
    padding-left: 40rpx;
    box-sizing: border-box;
}
.message-sent{
    flex-direction: row-reverse;
    color: #fff;
    display: flex;
}
.message-image{
    width:200rpx;
    height: 240rpx;
    padding-top:15rpx;
}
.message-input{
    box-sizing: border-box;
    width: 100%;
    height: 100%;
    border: 1rpx solid #c8c8cd;
    border-radius: 6rpx;
    background: #fff;
    padding: 0 10rpx;
    font-size: 30rpx;
}
```

样式 box-sizing:border-box 使容器具有一个规定大小的边框。

-webkit-mask-box-image 属性是为容器创建一个同样大小的遮罩，我们在视图层看到的图 6-10 所示的消息框效果便是基于这个属性来实现的。

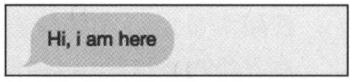


图 6-10

由于发送者与接收者消息框的样式不同，所以笔者分别声明了 message-text 与 message-sent-text 这两个不同的样式，然后在视图层上通过三目运算符判断 message 的 from 属性，渲染不同的样式。

在 wxss 的样式定义中，url 既可以接收一个图片地址，也可以接收一段直接的 image 数据。一些特定的 svg 数据，如上面的边框效果，并非人工直接输出的，需要先由矢量软件可视化创作，然后转成 svg 数据。对于小程序的初学者来说，对此不必过多深入。

6.3.5 调用图灵接口

在文档树中打开 pages/index/message.js 文件，将内容替换为如下 js 代码：

```
let app = getApp()
Page({
    data: {
        messages: [],
        inputContent: ''
    },
    onLoad(options) {
        let name = options.name || '聊聊'
        wx.setNavigationBarTitle({
        title: name
    })
        app.request(`http://1265839903.debug.open.weixin.qq.com/server/message.
            json`).then(res => {
            console.log(res.data)
            this.setData({
                messages: res.data
            })
        })
    },
    bindChange(e) {
        this.data.inputContent = e.detail.value
    },
    getTulingMsg(info) {
        let url = 'http://www.tuling123.com/openapi/api?key=88c5ff481e654bb68199
           82f60d288a97&info=${info}';
        app.request(url).then(res => {
            console.log(res)
            if (res.code == 100000) {
                let message = {
                    from: 'received',
                    text: res.text
                }
```

```
                this.setData({
                    messages: [...this.data.messages, message]
                })
            }
        })
    },
    sendMessage() {
        if (this.data.inputContent) return
        let message = {
            text: this.data.inputContent,
            from: 'sent'
        }
        this.setData({
            messages: [...this.data.messages, message],
            inputContent: ''
        })
        this.getTulingMsg(message.text)
    },
    previewImage(event) {
        wx.previewImage({
            urls: [event.target.dataset.image]
        })
    }
})
```

下面是针对上述代码的说明。

1）在函数 getTulingMsg 中，调用了网站 www.tuling123.com 的免费聊天接口，读者可自行申请。用于测试的 apikey，每天限量请求 5000 次。

2）函数 previewImage 用于预览消息中出现的图片，它调用了微信小程序图像预览接口 wx.previewImage。

3）当用户在视图层单击“发送”按钮时，会触发函数 sendMessage。在这个函数中，首先会判断 inputContent 是否为空，如果是，则返回。其次，新建一个 message 对象，将之推入 messages 数组，并与 inputContent 变量一起使用 setData 刷入视图层。最后，调用函数 getTulingMsg 取得自动应答的内容。

6.3.6 js 中的逻辑或操作

在周期函数 onLoad 中，下面这行代码：

```
let name = options.name || '聊聊'
```

表示如果 options.name 有值，则赋值于变量 name，否则使用默认值“聊聊”。“||”在 js 语言中是逻辑或操作符，用于在赋值语句中优先返回前面的值。

6.3.7 js 中的 let 关键字

let 是 js 语言中的块级作用域声明符，与 var 不同的是，它声明的变量仅在当前代码块

中有效。对于代码块，可以简单理解为一个花括号之内的范围，比如，一个函数、一个 if 语句、一个 for 语句均是块级作用域。

在开发中，let 和 var 关键字的异同如下：

1）声明后未赋值，表现相同。

```
(_ => {
    var varTest;
    let letTest;
    console.log(varTest); //输出undefined
    console.log(letTest); //输出undefined
})()
```

2）使用未声明的变量，表现不同。

```
(_ => {
    console.log(varTest) //输出undefined
    console.log(letTest) //直接报错
    var varTest = 'var1'
    let letTest = 'let1'
})()
```

3）重复声明同一个变量时，表现不同。

```
(_ => {
    var varTest = 'test var OK.';
    let letTest = 'test let OK.';
    var varTest = 'varTest changed.'
    //没问题
    let letTest = 'letTest changed.'
    //直接报错，提示变量已经声明过
})()
```

4）变量作用范围，表现不同。

```
(_ => {
    var varTest = 'test var OK.';
    let letTest = 'test let OK.';
    {
        var varTest = 'varTest changed.';
        let letTest = 'letTest changed.';
    }
    console.log(varTest)
    //输出"varTest changed.", 内部"{}"中声明的varTest变量覆盖了外部的letTest声明
    console.log(letTest)
    //输出"test let OK.", 内部"{}"中声明的letTest和外部的letTest不是同一个变量
    })()
```

从以上的对比中不难发现，let 关键字增强代码的安全性。在开发中，要优先、默认使用 let 关键字。

6.4 实现 my 页面

使用本书 2.1.3 节快捷创建页面的方法，创建 pages/index/my 页面。

在文档树中打开 pages/index/my.wxml 页面，将内容替换为如下标签代码：

```
<view class="weui-panel">
    <view class="weui-panel__bd">
        <view class="weui-media-box weui-media-box_appmsg" hover-class="weui-
            cell_active">
            <view class="weui-media-box__hd weui-media-box__hd_in-appmsg">
                <image class="weui-media-box__thumb" src="{{userInfo.avatarUrl}}" />
            </view>
            <view class="weui-media-box__bd weui-media-box__bd_in-appmsg">
                <view class="weui-media-box__title">{{userInfo.nickName}}</view>
                <view class="weui-media-box__desc">{{userInfo.city}}</view>
            </view>
        </view>
    </view>
</view>
<view class="weui-cells">
    <navigator url="about" class="weui-cell weui-cell_access">
        <view class="weui-cell__hd">
            关于
        </view>
        <view class="weui-cell__bd weui-cell__ft_in-access"></view>
    </navigator>
</view>
```

用户对象 userInfo 将在逻辑层获取。相对页面 about 尚不存在，稍后再定义。

使用 app.getUserInfo 拉取用户对象

在文档树中打开 pages/index/my.js 文件，将内容替换为如下代码：

```
Page({
    data:{
        userInfo: {}
    },
    onLoad() {
        let app = getApp()
        app.getUserInfo().then(userInfo => {
            this.setData({
                userInfo: userInfo
            })
        })
    }
})
```

sim.js 类库中的函数 app.getUserInfo，用于拉取当前用户对象，并返回一个 Promise。

在文档树中打开 sim.js/index.js 文件，可以找到函数 getUserInfo，代码如下：

```
function getUserInfo() {
    return new Promise(function (resolve, reject) {
        if (app.data.userInfo) {
            resolve(app.data.userInfo)
            return
        }

        wx.showNavigationBarLoading()
        let complete = function () {
            wx.hideNavigationBarLoading()
        }

        wx.login({
            success: _ => {
                wx.getUserInfo({
                    success: res => {
                        app.data.userInfo = res.userInfo
                        resolve(app.data.userInfo)
                    },
                    fail: reject,
                    complete: complete
                })
            },
            fail: reject,
            complete:complete
        })
    })
}
```

在这个函数中，首先是检查在 app.data 中是否已经存在 userInfo，如果有，则直接返回。其次，调用小程序的用户登录接口 wx.login，成功之后再调用拉取用户信息的接口 wx.getUserInfo。将获取到的 userInfo 存储在 app.data，以便下次拉取时直接使用。

6.5 实现 about 页面

使用本书 2.1.3 节快捷创建页面的方法，创建 pages/index/about 页面。

打开 pages/index/about.wxml 文件，替换为如下标签代码：

```
<view class="weui-msg">
    <view class="weui-msg__icon-area">
        <image src="/static/image/logo.png" style="width:170rpx;height:
            170rpx;border-radius:10rpx;"></image>
    </view>
    <view class="weui-msg__text-area">
        <view class="weui-msg__title">图灵聊聊</view>
        <view class="weui-msg__desc">基于图灵接口实现的自动聊天示例</view>
    </view>
    <view class="weui-msg__extra-area">
```

```
        <view class="weui-footer">
            <view class="weui-footer__links">
                <navigator class="weui-footer__link">www.rixingyike.com</navigator>
            </view>
            <view class="weui-footer__text">日行一刻 Copyright ? 2017</view>
        </view>
    </view>
</view>
```

这个页面没有变量绑定，没有逻辑层代码。实现的是一个关于页面，有 Logo、简介、网址等，适合用作一般的项目介绍页。

6.6 下载源码

在本节的学习过程中，如果遇到问题，可以通过如下方式来解决。

- 加入小程序微信群与作者及其他读者一同探讨。微信扫描前言中的二维码，关注“艺述思维”，发送“小程序”进群。
- 下载作者的源码，对照查找问题。在公众号中发送“图灵聊聊 6.5”获取下载地址。

第 7 章 Chapter 7

豆豆电影服务端

在前面第 2 章的示例中，使用的豆瓣接口存在每个 IP 每分钟只能调用 10 次的限制，如果刷久了，特别是使用即时搜索功能的时候，就会出现接口调用失败的情况。

本章将以 Golang 语言为基础，建立一个服务端，为豆豆电影提供稳定的接口服务，去除限制。

Golang 创建自谷歌，目前已是一门成熟的、开源的、类库完备的高级编程语言，用该语言既可“屠牛”，也可“杀鸡”。笔者认为，它是与前端小程序开发最配套的后端语言。

初学者学习编程，若前端语言学习 JS，后端语言学习 Golang，则无往而不利。

7.1 开发后端程序

7.1.1 安装 Golang 语言包

Golang 语言，也称 Go 语言。请前往谷歌 Golang 官网 https://golang.org/dl/，下载 Golang 安装包，如图 7-1 所示。

图 7-1

然后选择与所用电脑系统相适配的版本，根据提示完成安装。

截至本书结稿时，Golang 最新的稳定版本为 1.8.3。它具有良好的向前兼容性，笔者自 Golang1.1 版本开始使用，很多旧代码现在仍可以正常维护。

读者若无法从官网下载，则可在微信公号“艺述思维”中回复“Go 语言安装包下载”，获取国内的镜像地址。

7.1.2 安装仓库管理工具 git

在安装好 Golang 之后，打开网址 https://git-scm.com/downloads，下载 git 并安装，如图 7-2 所示。

图 7-2

截至本书发稿时，最新的 git 稳定版本是 2.13.2，下载与自己系统相适配的版本，根据提示安装即可。

Golang 安装类库使用的是 go get 指令，该指令需要远程从 github 网站上下载源码，依赖于 git 指令，安装 git 工具便是为了满足 go get 指令所需。另外，很多 Golang 类库在国内都无法访问，这也会导致 go get 执行失败。

在装有 Mac、Linux 系统的电脑上，有的终端环境可以直接运行指令。在 Windows 电脑上，安装 git 工具之后，会同时安装一个友好易用的 git 命令行工具，以方便执行 go get 指令，这也是在 Windows 上安装 git 工具的另外一个作用。

指令 go env 用于打开 Golang 语言环境变量，可以用它体验环境变量的设置是否正确。

7.1.3 安装 Go 语言编辑器

写代码怎么可以没有一个称手的编辑器？LiteIDE 是一款简单、开源免费、跨平台的 IDE，可从以下地址下载：

https://golangtc.com/download/liteide

LiteIDE 的安装很简单，按照提示进行操作即可。另外，被称为 ws 的 WebStorm 是更为强大的综合编程软件，但它是付费的，感兴趣的读者可以自行搜索、下载试用。

7.1.4 使用 sim.go 类库

打开命令行工具，输入以下指令：

```
go get github.com/rixingyike/sim.go
```

这行指令是为了安装 sim.go 所依赖的第三方类库。找到第 2 章创建的豆豆电影项目的根目录，创建一个 server 目录，然后在命令行下执行 cd 命令到该目录，并输入如下指令：

```
git clone https://github.com/rixingyike/sim.go.git ./
```

这行命令是为了下载 sim.js 类库，以便在此基础之上建立豆豆电影服务端。sim.js 是笔者开发的一个免费、开源的第三方类库，旨在帮助初学者快捷建立自己的小程序后端程序。

打开命令行工具，输入以下指令：

```
go get github.com/codegangsta/gin
```

该指令是为了安装编译工具 gin。go 语言是编译型语言，它与解析型语言（如 JS、Ruby、Python、Perl 等）最大的不同之处在于每次修改完代码之后都需要将代码重新编译成二进制文件才可以执行。gin 工具是用 Go 语言编写的，安装 gin 工具是为了在开发过程中实时监控代码、自动编译并刷新执行，这样编译型语言相对于解析型语言在开发中的不便就大大降低了。

现在，回到 server 目录，在命令行中执行如下脚本：

```
./debug.sh
```

该脚本的内容是：

```
gin -a 4001 -p 4000 run main.go
```

debug.sh 脚本是为了在 4000 端口启动服务端程序。执行完毕后，在浏览器中打开 http://localhost:4000/hi，如果看到“hi,sim.go”输出，则说明启动成功了。

7.1.5　创建豆瓣接口

打开 server/controller 目录，新建一个 douban.go 文件，将内容替换为如下代码：

```
package controller

import (
    "gopkg.in/kataras/iris.v6"
    "../lib"
    "fmt"
)

const DOUBAN_API_BASE = "https://api.douban.com/v2"
var apiResultCache = make(map[string]string)

func init() {
    // 拉取单个电影信息
    web.Get("/movie/subject/:id", func(c *iris.Context) {
        var id = c.Param("id")
        var pathKey = fmt.Sprintf("movie/subject/%s",id)

        if result, exist := apiResultCache[pathKey]; exist {
            c.WriteString(result)
            return
        }

        var result = sim.HttpGet(fmt.Sprintf("%s/%s",DOUBAN_API_BASE,pathKey),
          nil,nil)
        apiResultCache[pathKey] = result
        c.WriteString(result)
    })

    // 拉取三个榜单数据
```

```
    web.Get("/movie/bang/:key", func(c *iris.Context) {
        var key = c.Param("key")
        var pathKey = fmt.Sprintf("movie/%s",key)

        if result, exist := apiResultCache[pathKey]; exist {
            c.WriteString(result)
            return
        }

        var result = sim.HttpGet(fmt.Sprintf("%s/%s",DOUBAN_API_BASE,pathKey),
           nil,nil)
        apiResultCache[pathKey] = result
        c.WriteString(result)
    })

    // 搜索
    web.Get("/movie/search", func(c *iris.Context) {
        var q = c.URLParam("q")
        var pathKey = fmt.Sprintf("movie/search/%s",q)

        if result, exist := apiResultCache[pathKey]; exist {
            c.WriteString(result)
            return
        }

        var result = sim.HttpGet(fmt.Sprintf("%s/movie/search?q=%s",DOUBAN_API_BASE,q),
           nil,nil)
        apiResultCache[pathKey] = result
        c.WriteString(result)
    })
}
```

下面对上述代码进行说明。

1）第一行代码 package controller 用于定义包名为 controller。为了便于维护，包名要与目录同名。每个 Go 程序都是由包组成的，关于包的概念及其练习代码，参见本书的 17.1.2 节。

2）import 关键字引入了三个类库，第一个类库 gopkg.in/kataras/iris.v6 是 iris web 引擎，该文件用到了它的 iris.Context。第二个类库 ../lib 引用的是 sim.go 类库的 lib 目录。第三个类库 fmt 是 Golang 官方类库，它是处理字符串打印与格式化的类库。关于 import 关键字引入包的更多内容，参见本书的 17.1.2 节。

3）import 关键字引入类库有两种方式，一是基于类库名，如上面的第一个和第三个引入便是；二是基于相对目录，如上面的第二个引入便是。

4）严格来讲，sim.go 并不是一个纯粹的类库，而是一个帮助初学者快速建立后台程序的工具框架，其 lib 目录是框架的核心，用于提供常用的工具类方法及 Web 站点所需的核心功能。

下面这行代码：

```
const DOUBAN_API_BASE = "https://api.douban.com/v2"
```

用于定义一个常量，const 是 Go 语言定义常量的关键字。

下面这行代码：

```
var apiResultCache = make(map[string]string)
```

表示使用 var 关键字定义一个变量，Go 语言的 var 关键字与 JS 中的 var 含义相同。关于 var 关键字及变量的更多内容，参见本书的 17.1.4 节。

make 是 Go 语言的关键字，其在这里的作用是新建一个字典实例。map[string]string 是键、值均为 string 的字典，string 是 Go 语言的字符串类型。关于字典类型 map，参见本书的 17.3.6 节。关于字符串，参见本书的 17.1.5 节基本类型。

文件中的函数 func init() 将会自动执行，这是 Go 语言的内在机制。在这个函数内建立接口逻辑，可以免去与其他部分的耦合。增加控制器，只需在 controller 目录下新增类似的 Go 语言文件；移除控制器，删除文件即可。

在函数 init 内，调用了三次 web.Get，分别声明了三个接口给小程序使用。Web 内嵌的站点引擎是在文件 servr/controller/web.go 中定义的。这里，三个接口的声明是类似的，在第一个接口“/movie/subject/:id”中，“:id”是路径参数，c.Param("id") 是获取其参数值。

fmt.Sprintf 是字符串格式化函数，“%s”表示接收一个文本作为参数，在上下文中即为 id。

下面这段代码：

```
if result, exist := apiResultCache[pathKey]; exist {
    c.WriteString(result)
    return
}
```

用于从字典 apiResultCache 中检查键名为 pathKey 的键值是否存在。在 Go 语言中，if 条件控制语句不使用括号“()”，这一点与 JS 不同。关于 if 语句，参见本书的 17.2.2 节。

c.WriteString 函数是输出字符串到本次 http 请求。

下面这行代码则表示调用 sim.HttpGet，请求豆瓣 Api：

```
var result = sim.HttpGet(fmt.Sprintf("%s/%s",DOUBAN_API_BASE,pathKey),nil,nil)
```

sim.HttpGet 有三个参数，第一个是 url 地址，为了简单起见，我们将不重要的页码、起始数都略去了。第二个参数是 url 参数，本请求示例没有该参数，所以传递 nil，nil 在 Go 语言中代表空对象。第三个参数是一个 header 字典，在本次请求示例中仍然传递 nil。

apiResultCache[pathKey] = result 这句代码，是将结果存进 apiResultCache 字典，以便下次直接取用。

接口“/movie/bang/:key”与接口“/movie/subject/:id”类似，bang 是为了避免路径冲

突而添加的。

在第三个接口“/movie/search”中，c.URLParam("q") 用于获取 url 参数 q，这也是与前面两个接口 c.Param("key") 不同的地方，除此以外，其他代码均与前面类似。

douban.go 文件完成编辑之后，在浏览器中访问 http://localhost:4000/movie/bang/in_theaters，如果有内容输出，则说明后端程序工作了。

使用 go 语言编写的后端程序，其自身即可提供强大有效的 Web 服务，性能不输于 Nginx、Apache、IIS 等 Web Server，无须再另外搭建环境。而且 Go 语言编译的二进制文件，对类库无依赖，在环境一致的系统上部署时，复制过来即可运行，这也给运维带来了极大的便利。相比之下，使用其他开发语言，例如 Ruby、Python、Node、Java 等，它们都有环境依赖，仅是配置这些环境就是一道不小的门槛，这也是建议初学者学习 Go 语言的原因之一。Go 语言对于初学者来说简单易学，已广泛应用于各行各业。

7.2 改写小程序前端

使用微信开发者工具，打开第 2 章的豆豆电影项目。在这个项目中，有 5 个地方调用了豆瓣接口，现在我们逐一把它们换成适才所写的本地接口。

在文档树中打开 pages/douban/splash.js 文件，将函数 onLoad 中的：

```
app.request("https://api.douban.com/v2/movie/coming_soon?start=0&count=3")
```

替换为：

```
app.request("http://localhost:4000/movie/bang/coming_soon?start=0&count=3")
```

将 pages/douban/search.js 文件中的：

```
app.request('https://api.douban.com/v2/movie/search?q=${this.data.searchWords}&s
    tart=${start}&count=${this.data.size}')
```

替换为：

```
app.request('http://localhost:4000/movie/search?q=${this.data.searchWords}&start=
    ${start}&count=${this.data.size}')
```

将 pages/douban/list.js 文件中的：

```
app.request('https://api.douban.com/v2/movie/${this.data.type}?start=${start}&count=
    ${this.data.size}')
```

替换为：

```
app.request('http://localhost:4000/movie/bang/${this.data.type}?start=${start}&count=
    ${this.data.size}')
```

将 pages/douban/item.js 文件中的：

```
app.request('https://api.douban.com/v2/movie/subject/${options.id}')
```

替换为：

```
app.request('http://localhost:4000/movie/subject/${options.id}')
```

将 pages/douban/index.js 文件中的：

```
app.request('https://api.douban.com/v2/movie/${board.key}?start=0&count=10')
```

替换为：

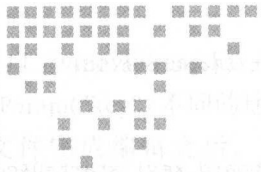
```
app.request('http://localhost:4000/movie/bang/${board.key}?start=0&count=10')
```

完成以上替换工作之后，项目已经不再直接依赖豆瓣服务器的接口了，在微信小程序后台管理页面中，可以将服务器域名“api.douban.com”移除了。

7.3　下载源码

在本节的学习过程中，如果遇到问题，可以通过如下方式来解决。

- 加入小程序微信群与作者及其他读者一同探讨。微信扫描前言的二维码，关注“艺述思维”，发送“小程序”进群。
- 下载作者的源码，对照查找问题。在公众号中发送“豆豆电影 7.2”获取下载地址。服务端的源码在小程序项目的 server 目录中。



Chapter 8 第 8 章

计算皮相服务端

没有数据库的后端（服务端），不是真正的后端。本章将为第 3 章的小程序项目“计算皮相”添加一个后端，实现分用户存储计算历史。

8.1 创建服务端程序

打开第 3 章介绍的小程序“计算皮相”，在根目录下创建一个 server 目录，然后在命令行下执行 cd 命令，进入该目录之后，输入如下指令：

```
git clone https://github.com/rixingyike/sim.go.git ./
```

之后执行 debug.sh 脚本，启动 Web 站点并开启热编译。

最后，在浏览器中打开 http://localhost:4000/hi，如果有内容输出，则说明站点启动成功。

8.1.1 启用 sqlite3 数据库与小程序服务端的自动登录功能

打开 server/config.ini 文件，其是站点的配置文件，找到 sqlite3 与 weapp 的配置段落，代码如下所示：

```
[sqlite3]
enable = true
filepath = "./sqlite3.db"

[weapp]
enable = true
app_id = "wxa7cec365b9bc44ca"
```

```
app_secret = "小程序密钥串"
```

将两个配置段落的 enable 均设置为 true，代表开启这个功能。修改 app_id 与 app_secret 为自己的小程序 id 和密钥。关于 app_id 与 app_secret，参见本书的 1.1 节，在微信小程序后台中查看。

在配置文件 config.ini 中，已开启了对 sqlite3 数据库的支持。sqlite3 是目前最为成熟的嵌入数据库，现已广泛应用于移动设备中。虽然在服务器端的开发中，一般受到追捧的是 MySQL、Oracle、MongoDB 等大型数据库，文件数据库 sqlite3 并不受青睐。但对于一般的起步应用来说，特别是对于初学者，运行起来才是最重要的。达尔文曾说过，在自然选择中被淘汰的不是最弱的，而是反应最慢的。sqlite3 数据库单文件达到 2GB 是没有问题的，对于小项目已然足够。况且同样是关系数据库，sim.go 对它的支持和对 MySQL 的支持是一样的，因此，可以在数据量上来之后，再无缝迁移到 MySQL 数据库上，代码无须做任何改动。

完成修改后重新执行 debug.sh 脚本，重启站点。

8.1.2　安装命令行工具 curl

curl 是 Linux 程序员广为熟知的文件传输工具，它利用 URL 语法在命令行方式下工作，也被广泛应用于接口测试。

如果读者的电脑是 Windows 系统，在 7.1.2 节安装 git 工具时，便自动安装了 curl 工具。如果是 Ubuntu 系统，则使用如下指令安装 curl：

```
sudo apt install curl
```

如果是 Mac 系统，则先执行下面的指令安装 Homebrew：

```
ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

Mac 系统自带 ruby 解释器，这行指令是在线安装 Homebrew 软件包管理器。完成后执行下面的命令：

```
brew install curl
```

即可安装 curl。

8.1.3　关于一般性通用接口的解读

打开 server/controller/blog.go 文件，这是一个实现简单数据库操作的示例控制器，开启数据库支持（无论是 sqlite3 还是 MySQL）之后，在调试状态下会自动向库中写入名为 blog 的数据表，并且会开启如下接口：

```
POST http://localhost:4000/blog
GET http://localhost:4000/blog/1
GET http://localhost:4000/blogs
```

```
DELETE http://localhost:4000/blog/1
PUT http://localhost:4000/blog/1
```

其中，POST、GET、DELETE、PUT 是 http 协议与服务器交互的四种不同方法，分别代表新建、请求、删除、更新。

然后，在命令行中执行如下指令：

```
curl -X POST -d '{"title":"x"}' http://localhost:4000/blog
```

这行指令表示向接口新增一条记录，其中字段 title 的值为 x。

再在命令行中输入如下指令：

```
curl http://localhost:4000/blog/1
```

这条指令表示拉取 id 为 1 的 blog 记录。

之后，在命令行中输入如下指令：

```
curl http://localhost:4000/blogs
```

这行指令表示拉取所有 blog 记录（限 1000 条以内），如果指定 page 与 size 参数，则是分页拉取，示例如下：

```
curl http://localhost:4000/blogs?page=1&size=10
```

这行指令表示以每页大小为 10，拉取第 1 页的记录。page 参数自 1 开始计数。

接着，在命令行中输入以下指令：

```
curl -X DELETE http://localhost:4000/blog/1
```

这行代码表示删除 id 为 1 的 blog 记录。

最后，在命令行中输入以下指令：

```
curl -X PUT -d '{"title":"xxx"}' http://localhost:4000/blog/1
```

这行代码表示修改 id 为 1 的 blog 记录，将 title 字段的值修改为 xxx。执行完这条 curl 指令之后，数据库中 id 为 1 的记录即完成了修改。

8.1.4 使用 SQLiteStudio

SQLiteStudio 是一款跨平台的 sqlite3 数据库文件管理工具，可以很方便地查看库表内容。在浏览器中打开 https://sqlitestudio.pl/index.rvt?act=download，如图 8-1 所示，即可下载安装该工具。

Distribution	Platform	Size	Version	Link
Windows	32-bit	16.4MB	3.1.1	sqlitestudio-3.1.1.zip
Linux	64-bit	18.7MB	3.1.1	sqlitestudio-3.1.1.tar.xz
MacOSX	64-bit (ix86_64)	25.2MB	3.1.1	sqlitestudio-3.1.1.dmg
Sources (zip)	Independent	9.0MB	3.1.1	sqlitestudio-3.1.1.zip
Sources (tar.gz)	Independent	8.3MB	3.1.1	sqlitestudio-3.1.1.tar.gz

图 8-1

打开 server/sqlite3.db 文件，选择 blog 表并生成一条查询语句，如图 8-2 所示。

单击“执行语句”按钮，就可以看到刚才使用 curl 工具创建的记录了，如图 8-3 所示。

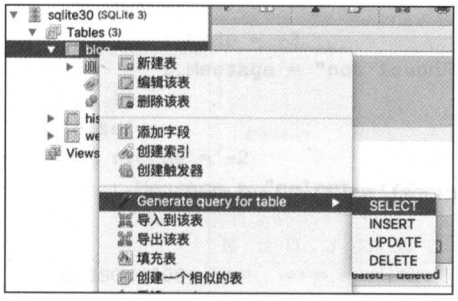


图　8-2

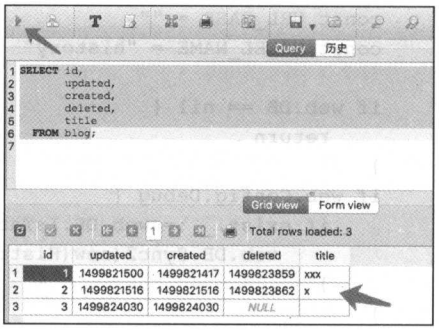


图　8-3

在系统的文件资源管理器中，直接将 sqlite3 文件拖进 SQLiteStudio，就可以快速新建链接。单击左上角的“连接到数据库”按钮，即可快速打开数据库，如图 8-4 所示。

图　8-4

8.1.5　扩展新的控制器

熟悉了 blog.go 文件之后，便能很容易地扩展新的数据模型。在 server/controller 目录之下，复制 blog.go 为 history.go。

打开 server/controller/history.go 文件，将内容修改为：

```
package controller

import (
    "../lib"
    "gopkg.in/kataras/iris.v6"
    "fmt"
    "math"
)

type (
    History struct {
        sim.Model `xorm:"extends"`
        UserId int64 `json:"user_id"`
        Record        string `xorm:"char(50)" json:"record"`
    }
    HistoryPage struct {
        List       []History `json:"list"`
        Size       int `json:"size"`
        Page       int `json:"page"`    //自1开始计数
        Count      int `json:"count"`
        TotalPage  int `json:"total_page"`
    }
)

func init() {
```

```
const URL_BASE = ""
const MODEL_NAME = "history"

if web.DB == nil {
    return
}
if web.Config.Debug {
    if exist,_ := web.DB.IsTableExist(MODEL_NAME); !exist {
        web.DB.Sync2(new(History))
    }
}

/*
使用curl测试接口
POST   /history     创建
curl -X POST -d '{"title":"x"}' http://localhost:4000/history

GET    /history/1 查看
curl http://localhost:4000/history/1

GET    /historys
curl http://localhost:4000/historys

DELETE /history/1 删除
curl -X DELETE http://localhost:4000/history/1

PUT    /history/1 更新
curl -X PUT -d '{"title":"xxx"}' http://localhost:4000/history/1
*/

var sub = web.Router
if URL_BASE != "" {
    sub = web.Party(URL_BASE)
}
// 拉取单条记录
sub.Get(fmt.Sprintf("/%s/:id", MODEL_NAME), func(c *iris.Context) {
    var r sim.Result

    var id, err = c.ParamInt64("id")
    if err != nil {
        r.Code = -1
        r.Message = "id unvalid"
        c.JSON(200, r)
        return
    }

    var data History
    if _, err := web.DB.ID(id).Get(&data); err == nil {
        if data.ID > 0 {
        r.Code = 1
        r.Message = "success"
```

```
            r.Data = data
        }else{
            r.Code = -3
            r.Message = "not found"
        }
    }else{
        r.Code = -2
        r.Message = "select err"
    }

    c.JSON(200, r)
})

// 分页拉取记录，如果没有指定size，则拉取所有记录（限1000条以内）
sub.Get(fmt.Sprintf("/%ss", MODEL_NAME), func(c *iris.Context) {
    var r sim.Result
    var page,_ = c.URLParamInt("page")
    var size,_ = c.URLParamInt("size")

    if page == 0 {
        page = 1
    }
    if size == 0 {
        size = 1000
    }
    var data = HistoryPage{Page:page,Size:size}
    var offset = (data.Page - 1) * data.Size
    var my = web.GetWeappUser(c)

    if err := web.DB.Where("user_id = ?", my.ID).Desc("id").Limit(data.Size,
        offset).Find(&data.List); err == nil {
        if total, err := web.DB.Where("user_id = ?", my.ID).Count(new(History));
            err == nil {
            data.Count = int(total)
            data.TotalPage = int(math.Ceil(float64(total) / float64(data.Size)))
        }else{
            sim.Debug("select count err", err.Error())
        }
        //for k,v := range data.List{
        // web.DB.GetOneById(&v.User,v.UserId)
        // data.List[k].User = v.User
        //}
        r.Code = 1
        r.Data = data
    }else{
        sim.Debug("select page err", err.Error())
    }

    c.JSON(200, r)
})
```

```
// 新增单条记录
sub.Post(fmt.Sprintf("/%s", MODEL_NAME), func(c *iris.Context) {
    var r sim.Result

    var one History
    if err := c.ReadJSON(&one); err != nil {
        sim.Debug("read data err", err.Error())
        r.Code = -1
        r.Message = "read data err"
        c.JSON(200, r)
        return
    }

    var my = web.GetWeappUser(c)
    one.UserId = my.ID

    if affected, err := web.DB.Insert(&one); err == nil {
        if affected > 0 {
            r.Code = 1
            r.Data = one.ID
        }
    }else{
        sim.Debug("post new bean err",err.Error())
    }

    c.JSON(200, r)
})

// 删除单条记录
sub.Delete(fmt.Sprintf("/%s/:id", MODEL_NAME), func(c *iris.Context) {
    var r sim.Result

    var id, err = c.ParamInt64("id")
    if err != nil {
        r.Code = -1
        r.Message = "id unvalid"
        c.JSON(200, r)
        return
    }

    var one History
    if _, err := web.DB.ID(id).Get(&one); err != nil {
        r.Code = -2
        r.Message = "not found"
        c.JSON(200, r)
        return
    }

    if affected, err := web.DB.Id(id).Delete(&one); err == nil {
        if affected > 0 {
            r.Code = 1
```

```
        }
    }

    c.JSON(200, r)
})

// 更新单条记录
sub.Put(fmt.Sprintf("/%s/:id", MODEL_NAME), func(c *iris.Context) {
    var r sim.Result

    var id, err = c.ParamInt64("id")
    if err != nil {
        r.Code = -1
        r.Message = "id unvalid"
        c.JSON(200, r)
        return
    }

    var one History
    if _, err := web.DB.ID(id).Get(&one); err != nil {
        r.Code = -2
        r.Message = "not found"
        c.JSON(200, r)
        return
    }

    if err := c.ReadJSON(&one); err != nil {
        r.Code = -3
        r.Message = "read data err"
        c.JSON(200, r)
        return
    }

    if _, err := web.DB.Id(id).Update(&one); err == nil {
        //如果当前内容与原内容一样,则affected会返回0
        r.Code = 1
        r.Data = one.ID
    }

    c.JSON(200, r)
})
}
```

history.go 文件与 blog.go 文件的大部分内容都是相同的，改动的是数据模型的定义，由原 Blog 改为 History。在结构体 History 中，原 Title 字段改为了 Record。一种简单的方法是将所有 Blog 替换为 History，不区别大小写，然后再逐项检查编辑器报错的地方。

一个数据库代表一块业务逻辑，如果逻辑是一样的，那么用同一个数据模型就可以了。使用复制粘贴的方法，可以快速扩展出新的数据模型，同时又可以最大限度地保持每一块业务逻辑的灵活性，随着功能的演化，各自的修改并不会影响其他的业务逻辑。

8.2 改写小程序前端

使用微信开发者工具，打开第 3 章的计算皮相项目，在文档树中打开 app.js 文件，于 app 的声明之下输入如下三行代码：

```
let app = require("./sim.js/index.js")
app.config.enableWeappUserAutoLogin = true
app.config.serverUrlBase = "http://localhost:4001"
```

其中，第二行代码将在 app.getUserInfo 函数中开启微信小程序用户自动登录服务器的功能，第三行代码是设置服务器接口地址的基地址。

8.2.1 使用 POST 方法新增数据

在 app.js 文件的 onLaunch 函数中，添加对 app.getUserInfo 函数的主动调用，拉取用户信息并自动登录到服务器，代码如下：

```
onLaunch: function() {
    app.getUserInfo()
}
```

在文档树中打开 pages/index/index.js 文件，添加如下代码：

```
postNewRecord(record) {
    let app = getApp()
    app.request('http://localhost:4000/history', { record: record }, { method:
        'POST' })
}
```

该函数向接口 http://localhost:4000/history 发起一个 POST 请求，并发送数据 { record: record }，这是数据库 history 新增记录需要用到的数据。

然后在函数 equalOperation 中，找到如下两行代码：

```
this.data.logs.push(data + " = " + result);
wx.setStorageSync("calclogs", this.data.logs);
```

将其替换为：

```
this.postNewRecord(data + " = " + result)
```

这里将由原来的在小程序本地缓存中存储记录，改为发送给服务器在服务端数据库中存储。

此外，在 pages/index/index.js 文件的 data 对象中，logs 数组已不再需要，可以删除了。

8.2.2 调用分页接口拉取数据

打开 server/controller/history.go 文件，找到“新增单条记录”处，往下看会发现有这样两行代码：

```
var my = web.GetWeappUser(c)
one.UserId = my.ID
```

其中，web.GetWeappUser(c) 用于获取当前登录的小程序用户对象，下一行代码即为将用户 ID 赋值给新记录的 UserId 字段。

在文档树中打开 pages/index/history.js 文件，将 onLoad 函数修改为：

```
onLoad(options) {
    let app = getApp()
    app.request('http://localhost:4000/historys').then(res => {
        this.setData({ "logs": res.data.list })
    })
}
```

新代码会向服务器的 /historys 接口发出请求，拉取至多 1000 条记录，然后将 history 列表赋值给页面变量 logs 数组。

在文档树中打开 pages/index/history.wxml，由于 logs 数组不再是字符串，而是对象数组，因此将列表渲染的标签代码修改为：

```
<view class="weui-cell" wx:for="{{logs}}" wx:key="id">
    <view class="weui-cell__bd">{{item.record}}</view>
</view>
```

上述代码实现了两种改动：一是修改 wx:key 为 id，id 是记录项主键，在列表渲染中可作为主键；二是修改原 item 为 item.record。

至此就实现了计算皮相项目的服务器端程序。每次计算的结果都将自动存入数据库中，单击“历史页面”，将会看到所有历史记录。

8.3　下载源码

在本节的学习过程中，如果遇到问题，可以通过如下方式来解决。

- 加入小程序微信群与作者及其他读者一同探讨。微信扫描前言中的二维码，关注“艺述思维”，发送“小程序”进群。
- 下载作者的源码，对照查找问题。在公众号中发送“计算皮相 8.2”获取下载地址。服务端的源码在小程序项目的 server 目录中。

Chapter 9 第 9 章

黑黑天气服务端

本章将为第 4 章介绍的小程序“黑黑天气”添加一个服务端，用于存储历史上查询过的天气。具体实现方式为：将原来使用 ip 查询城市的方法，修改为调用微信小程序获取地理位置的接口，先获取经纬度信息，再以经纬度信息调用谷歌地理接口获取城市名称，并新增一个 history 页面，用于展示查询过的天气记录。

在技术上，本章将主要介绍前端与后端在 JSON 数据交互、处理上的一些实用技巧，例如动态 JSON 数据的解析、使用字典类型等。前端与后端的数据交换格式，除了 JSON，还有 XML、SocketMessage 等，但以 JSON 最为简单、高效、易学，初学者通晓 JSON 这一个数据格式就足以应对所有情况了。

完成后的运行效果截图如图 9-1 所示。

图 9-1

9.1 创建服务端程序

本节将实现用 sim.go 创建一个 Web 接口站点，并启动该站点，提供给前端小程序使用。

打开第 4 章的项目黑黑天气，在根目录下创建一个 server 目录，在命令行下执行 cd 命令，转到该目录，输入如下指令：

```
git clone https://github.com/rixingyike/sim.go.git ./
```

然后参照 8.1.1 节，修改 config.init 文件中的 sqlite3 与 weapp 段落，代码如下所示：

```
[sqlite3]
enable = true
filepath = "./sqlite3.db"

[weapp]
enable = true
app_id = "wx7dca91c025113d48"
app_secret = "小程序密钥"
```

之后，开启 sqlite3 数据库与小程序用户自动登录服务器的功能（注意，请将代码中的“小程序密钥”替换为自己的密钥）。

执行 debug.sh 脚本，启动 Web 站点并开启热编译。

在浏览器中打开 http://localhost:4000/hi，如果有内容输出，则说明站点启动成功。

9.1.1　使用万能的 JSON 字段

本节将创建 History 数据模型，用于存储从小程序前端发送过来的历史数据。

将第 8 章项目中的 server/controller/history.go 文件复制到本章项目的 server/controller 目录下。打开 server/controller/history.go 文件，将数据模型 History 修改为：

```
History struct {
    sim.Model `xorm:"extends"`
    UserId int64 `json:"user_id"`
    Record map[string]string `xorm:"json" json:"record"`
}
```

这里与原 History 的差别不大，仅改动了两个地方。

1）修改字段 Record 的 Tag 声明，如上面的代码所示，设置“xorm”为“json”。也就是说，在存储时，先将本字段序列化为 JSON 字符串；取出时与之相反，先将 Record 字段反序列为 map 对象。这个序列化、反序列化的过程是由框架自动完成的。

2）将 Record 字段的数据类型，由字符串修改为字典，以便于存储丰富的结构化信息。

9.1.2　特改特定的接口逻辑

找到“新增单条记录”的接口，看到下面这两行代码：

```
var my = web.GetWeappUser(c)
one.UserId = my.ID
```

之后添加：

```
//检查当天的天气是否已经记录过
var now = time.Now()
var todayStartTime = time.Date(now.Year(),now.Month(),now.Day(),0,0,0,0,now.
   Location())
```

```
var tomorrowStartTime = todayStartTime.AddDate(0,0,1)
if exist,err := web.DB.Where ("created < ? and created > ? and user_id = ?",
    tomorrowStartTime, todayStartTime, my.ID).Get(&one); err == nil && exist {
    //如果已存在当天记录
    r.Code = -2
    r.Message = "record exist"
    c.JSON(200, r)
    return
}
```

这段代码首先获取当前时间，time.Now()返回当前时间，然后根据当前时间，算出今日之零点时间todayStartTime，接着使用AddDate方法添加1天时间，算出tomorrowStartTime，最后检查在这个时间段之内有无记录，并且要求user_id等于当前用户ID。

从上面的代码可以看出，笔者可以独立地修改每个控制器中每个接口的代码逻辑，这是sim.go框架的便利之一，各个控制器处于完全松耦合的状态，在程序升级和改造时可以最大限度地避免对其他部分产生影响。

9.1.3 解析动态JSON数据的方法

本节将展示在Go语言开发中，如何解析动态的JSON数据。在Go语言中，解析JSON数据的基础方法是，首先声明一个与目标数据结构相同的结构体，然后使用json.Unmarshal方法将数据反序列化至一个结构体实例中，以此来实现对JSON数据的解析。基础的解析方法如下面的代码所示：

```
type Student struct {
    Name    string
    Age     int
    Guake   bool
    Classes []string
    Price   float32
}
var jsonStr = `{"Name":"Weapp","Age":16,"Guake":true,"Classes":["Math","English"],
   "Price":9.99}`
var student Student
json.Unmarshal([]byte(jsonStr), &student)
```

不过，基础的解析方法仅在明确知道目标数据结构的前提下适用，下面笔者示范一下在不知道目标数据的结构（即目标数据是动态JSON）的情况下如何实现解析。

在server/controller目录下，添加city.go文件，代码如下：

```
import (
"../lib"
"gopkg.in/kataras/iris.v6"
"fmt"
)
```

```
func init() {
    web.Get("/city", func(c *iris.Context) {
    var r sim.Result
    var longitude = c.URLParam("lng")
    var latitude = c.URLParam("lat")

    var apiUrl = fmt.Sprintf("http://maps.google.cn/maps/api/geocode/json?latlng
       =%s,%s&language=CN",latitude,longitude)
    var jsonResult = sim.HttpGet(apiUrl,nil,nil)

    if doc,err := sim.ParseJsonToDocument(jsonResult); err == nil {
        if results,err := doc.Path("results").Children();err == nil && len(results)>0 {
            var result = results[0]
            var addresses,_ = result.Path("address_components").Children()
            for _,address := range addresses {
                var types,_ = address.Path("types").Children()
                if types[0].Data().(string) == "locality" {
                    r.Code =1
                    r.Data = address.Path("long_name").Data().(string)
                }
            }
        }
    }
    c.JSON(200, r)
    })
}
```

在 controller 目录下，每一个文件定义一个控制器，控制一段路由。city.go 文件扩展了一个新的控制器，这个控制器仅包括一个接口“/city”。在这个接口中，首先通过 c.URLParam 获取经纬度参数，然后调用谷歌的地理逆向接口，传入经纬值，从而得到城市信息。

从谷歌的地理接口中返回的数据是这样的：

```
{
    "results" : [
        {
            "address_components" : [
                {
                    "long_name" : "格尔木市",
                    "short_name" : "格尔木市",
                    "types" : [ "political", "sublocality", "sublocality_level_1" ]
                },
                {
                    "long_name" : "海西蒙古族藏族自治州",
                    "short_name" : "海西蒙古族藏族自治州",
                    "types" : [ "locality", "political" ]
                },
                {
                    "long_name" : "青海省",
```

```
                "short_name" : "青海省",
                "types" : [ "administrative_area_level_1", "political" ]
            },
            {
                "long_name" : "中国",
                "short_name" : "CN",
                "types" : [ "country", "political" ]
            }
        ],
        "formatted_address" : "中国青海省海西蒙古族藏族自治州格尔木市",
        "geometry" : {...},
        "place_id" : "ChIJ2YYhWlaPmDcRUDYWZYvs_-Q",
        "types" : [ "political", "sublocality", "sublocality_level_1" ]
    },
    ...
],
"status" : "OK"
}
```

在浏览器中打开 http://maps.google.cn/maps/api/geocode/json?latlng=36,92&language=CN，可以看到以上结果。

然后在谷歌接口返回的 JSON 内容中，获取到州、市级别的城市名称，即 types 数组中第一个元素为 locality 的数据。

为了方便逐级查询节点，避免定义结构体的麻烦，可使用 sim.ParseJsonToDocument 工具方法将 JSON 内容解析为一个树状文档对象。该文档对象的 Path 方法会返回指定路径的节点，可以这样使用：

```
doc.Path("outter.inner.value1")
```

函数 Children() 返回节点的数组形式，例如 result.Path("address_components").Children() 将返回 address_components 节点的 4 个子节点。

需要特别指出的是，代码“types[0].Data().(string)”返回的是字面值，但“types[0].String()”返回的却是带双引号的字面值。前者代表取其值，其值是字符串类型；后者是将节点转换为字符串格式。

如上所示，本节通过 sim.go 类库自带的 sim.ParseJsonToDocument 方法实现了动态 JSON 数据的解析。sim.ParseJsonToDocument 方法是基于第三方开源类库实现的，直接使用该方法，可以避免引用类库的麻烦。

9.2 改写小程序前端

本节将调用在 9.1 节实现的后端接口，实现对历史记录的保存。

使用微信开发者工具打开第 4 章的小程序项目黑黑天气，在文档树中打开 app.js 文件，在 app 变量声明之下添加如下两行代码：

```
let app = require("./sim.js/index.js")
app.config.enableWeappUserAutoLogin = true
app.config.serverUrlBase = "http://localhost:4001"
```

在小程序端开启用户自动登录服务器的功能，并且设置服务器接口的基地址。如果将程序部署到外网，那么这个地址需要修改为线上地址。

9.2.1　使用不同的模拟器测试项目

如果使用 iPhone5 的模拟器测试该项目，会发现主要温度的 UI 与底边靠得太近，如图 9-2 箭头所示。

但这个问题在 iPhone6 中就不存在，这也说明程序在上线前需要在多种尺寸下进行测试。

要解决上述问题，可采用如下方式。在文档树中打开 pages/index/index.wxml 文件，找到文本“34℃”所在的 view 视图，修改它的 style 属性，如下所示：

```
<view class="weui-flex" style="display: flex;align-items: center; position:absolut
    e;bottom:0;width:100%;">
    <view style="font-size:120rpx" class="weui-flex__item">
        {{weather.wendu}}° C
    </view>
    <navigator url="history" style="text-align: center" class="weui-flex__item">
        {{weather.today.week}} {{weather.city}}
    </navigator>
</view>
```

粗体部分是新增的样式代码。使用相对于父容器的绝对位置进行布局，距父容器底部为 0 并将宽度设置为 100%。完成后瑕疵已经去除，测试结果如图 9-3 所示。

图　9-2

图　9-3

修改星期与城市文本的组件类型为 navigator，并将 url 指向 history 页面。history 页面

将显示个人查看天气的历史记录。

9.2.2 使用默认的页面数据避免渲染错误

在小程序测试过程中，发现 Console 面板会报告图 9-4 所示的错误。

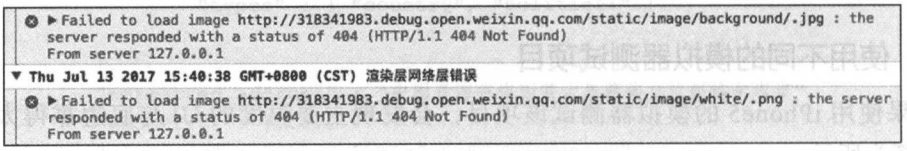


图 9-4

这是由于在 pages/index/index.wxml 文件中，绑定了变量 {{weather.today.typeBackgorund}} 与 {{weather.today.type}}，代码如下：

```
background-image: url('/static/image/background/{{weather.today.typeBackgorund}}.jpg')
    ...
    src="/static/image/white/{{weather.today.type}}.png"
```

但是在程序运行伊始，这两个变量并不存在，在天气数据正确加载之前，页面已经发生了渲染。最简单的解决办法是修改页面初始数据对象 data，并在对象 weather 里添加一个默认的 today 变量，代码如下：

```
weather: {
    today: { type: "多云", typeBackgorund: "default" }
}
```

其中，粗体部分是新增的代码。再次刷新项目，错误已不存在。

9.2.3 分离代码逻辑提高可阅读性

当一个函数的行数过多时，一般需要对它进行拆分，以保持代码的可阅读性和可维护性。代码是给机器用的，却是给程序员读的。不要因为一时的懒惰或耍酷而将代码写得难于理解。有些当时看起来很酷的代码，过几个月之后可能连自己都看不懂了。最好的代码，是一眼就能看明白的代码。本节正好会涉及一个行数过多的函数，笔者将示范如何依据其业务逻辑将之分离为不同的函数。

在文档树中打开 pages/index/index.js 文件，在 onLoad 函数上方新增 postTodayWeather 函数，代码如下：

```
postTodayWeather(record) {
    let app = getApp()
    app.request('http://localhost:4000/history', { record: record }, { method:
        'POST' })
}
```

该函数负责将一条新的查看记录保存到服务器，是数据新增操作，其与第 8 章计算皮

相保存计算记录的函数类似（参见本书 8.2.1 节的第 2 段代码），可将该段代码复制过来，在此基础之上进行修改。

所有新增的函数，默认放在函数区顶部。修改之后的 loadWeather 函数如下所示：

```
loadWeather() {
    let app = getApp()

    const background = {
        "大雨": "dayu",
        "中雨": "dayu",
        "小雨": "xiaoyu",
        "暴雨": "dayu",
        "雷阵雨": "leizhenyu",
        "晴": "qing"
    };
    // 查询地理位置
    let requestLocation = _ => {
        wx.getLocation({
            type: 'wgs84',
            success: function (res) {
                var latitude = res.latitude
                var longitude = res.longitude
                requestCity(latitude, longitude)
            }
        })
    }
    // let requestCity = app.request('http://int.dpool.sina.com.cn/iplookup/iplookup.
        php?format=json')
        // 根据地理信息查询城市
        let requestCity = (latitude, longitude) => {
            wx.showLoading({ title: "加载中" })
            let requestCity = app.request(`http://localhost:4000/city?lat=${latitude}
                &lng=${longitude}`).then(res => {
                var city = res.data
                requestWeather(city)
            }).catch(() => { wx.hideLoading() })
        }
        // 根据城市查询天气
        let requestWeather = (city) => {
            var weather = {};
            weather.city = city;
            app.request('http://wthrcdn.etouch.cn/weather_mini?city=${city}').then
                (res => {
                console.log(res)
                if (res.status === 1000) {
                    var data = res.data;
                    var forecast = data.forecast;       //天气特征与未来天气
                    var futureList = [];
                    for (var i = 0; i < forecast.length; i++) {
                        var one = forecast[i];
```

```
                var future = {
                    week: one.date.slice(-3),
                    type: one.type,
                    wendu: one.low.split(' ')[1] + "-" + one.high.split(' ')
                        [1]
                }
                futureList.push(future);
            }

            var today = futureList[0];
            console.log("type", background[today.type])
            if (background[today.type]) {
            today.typeBackgorund = background[today.type];
        } else {
            today.typeBackgorund = "default";
        }
        weather.wendu = data.wendu
        weather.today = today
        weather.futureList = futureList.slice(1);
        this.setData({
            weather: weather
        });
        this.postTodayWeather(today)
    }
    wx.hideLoading()
}).catch(() => { wx.hideLoading() })
}
requestLocation()
}
```

原函数的逻辑分为两步：先根据 ip 地址查询所在城市，再以城市查询天气信息。

新函数将逻辑扩展为了三步：首先调用小程序的地理接口 wx.getLocation，获取用户所在地区的经纬度；其次以经纬度信息调用 9.1 节新增的服务端接口获取城市名称；最后再以城市名称拉取天气信息。

两个函数最后一步的逻辑是相同的，可以保持代码不变。另外，为了使代码便于阅读和维护，可将新代码的三步逻辑分别定义为如下三个命名函数：requestLocation、requestCity、requestWeather。这三个函数均定义于函数之中，相当于三个类型为函数的变量，可以像普通函数一样调用。

由于数据格式修改，原代码

```
var city = res.city
```

改为了：

```
var city = res.data
```

此外，在函数 requestWeather 内部，添加了对新定义函数 postTodayWeather 的调用：

```
...
this.setData({
    weather: weather
});
this.postTodayWeather(today)
```

以此实现查看记录的保存。改写后的代码更易阅读。

9.2.4　在 WXML 页面中直接绑定字典数据

从服务器返回的字典类型的数据，在小程序前端是 Object 对象，因此不需要做任何额外的工作，即可直接在视图中绑定对象的属性，这为开发带来了很大的便利。

使用 2.1.3 节介绍的快捷创建页面的方法，创建 pages/index/history 页面。在文档树中打开 pages/index/history.wxml 文件，将标签代码替换为：

```
<view class="weui-cells">
    <view wx:for="{{histories}}" wx:key="id" class="weui-cell">
        <view class="weui-cell__bd">{{item.record.wendu}},{{item.record.type}}</
            view>
        <view class="weui-cell__ft">{{item.record.week}}</view>
    </view>
</view>
```

这里会涉及在 9.1.1 节新增的数据模型 History，其 Record 字段是 map 类型，但仍然可以像使用属性一样使用其键值，而不必定义太多字段，免去了麻烦。

在文档树中打开 pages/index/history.js 文件，将内容替换为如下代码：

```
Page({
    data: {
        histories:[]
    },
    onLoad(options) {
        let app = getApp()
        app.request('http://localhost:4000/historys').then(res => {
            console.log(res)
            this.setData({ "histories": res.data.list })
        })
    }
})
```

该页的代码与第 8 章的项目计算皮相中 pages/index/history.js 文件中的代码类似，可以复制过来在其基础上进行修改。

在文档树中打开 pages/index/history.json 文件，将页面标题修改为“查询历史”，代码如下所示：

```
{
    "navigationBarTitleText": "查询历史"
}
```

至此，所有前端、后端的代码都已经完成了。刷新项目，就可以看到与笔者截图（图 9-1）相同的运行效果。

9.3 下载源码

在本节的学习过程中，如果遇到问题，可以通过如下方式来解决。

- 加入小程序微信群与作者及其他读者一同探讨。微信扫描前言中的二维码，关注“艺述思维”，发送“小程序”进群。
- 下载作者的源码，对照查找问题。在公众号中发送“黑黑天气 9.2”获取下载地址。服务端的源码在小程序项目的 server 目录中。

第 10 章 Chapter 10

笑林百家服务端

七牛云存储是国内 Go 语言大牛许式伟创立的公司，新人注册即享有 10GB 的免费存储空间，对初学者来说完全够用。使用云存储，可以在提升资源访问速度的同时有效减少运维成本的投入，尤其适用于创业公司、中小企业，更适用于个人。

本章将为第 5 章介绍的小程序“笑林百家”添加一个顶部导航栏，并调用七牛云存储的 API 实现图片上传。完成后的效果如图 10-1 所示。

图 10-1

10.1 创建服务端程序

本节会为小程序“笑林百家”创建一个后端程序，辅助前端将图片上传至七牛云存储。不通过服务器中转，而是由客户端直接上传，可以减少服务器的压力和带宽的占用，从而有效提高整体的用户体验。此外，涉及密钥等机密信息也不适宜存储在前端，只适合存储在服务端。在项目中使用其他第三方类库时，应采用同样的策略。

打开“笑林百家”小程序，在根目录下创建一个 server 目录，在命令行下执行 cd 命令，转到该目录下，输入并执行如下指令：

```
git clone https://github.com/rixingyike/sim.go.git ./
```

然后参照 8.1.1 节修改 config.init 文件中 sqlite3 与 weapp 的代码段落，如下所示：

```
[sqlite3]
enable = true
filepath = "./sqlite3.db"

[weapp]
enable = true
app_id = "wx7dca91c025113d48"
app_secret = "小程序密钥"

[qiniu]
enable = true
scope = "七牛空间名"
access_key = "七牛ACCESS_KEY"
secret_key = "七牛SECRET_KEY"
watermark = ""
server_base = ""
```

开启 sqlite3 数据库，与小程序用户自动登录服务器的功能。注意，请将代码中粗体的“小程序密钥”替换为自己的密钥。

10.1.1 启用七牛云上传功能

在配置文件的 qiniu 段落中，将 enable 设置为 true，代表开启七牛云上传。而 scope、access_key 和 secret_key 则是实现上传功能的必需字段。

配置文件中的 qiniu → server_base 是生产环境用到的七牛空间的个性域名，在开发环境中默认会取用七牛的测试域名 7xndm1.com1.z0.glb.clouddn.com。

10.1.2 注册七牛账号与创建存储空间

本节将主要介绍七牛网站的使用，为使用七牛上传接口准备必要的信息。

在浏览器中打开 https://www.qiniu.com，按提示注册并登录。首先，打开对象存储链接 https://portal.qiniu.com/bucket，单击“新建存储空间”按钮，出现图 10-2 所示的界面。

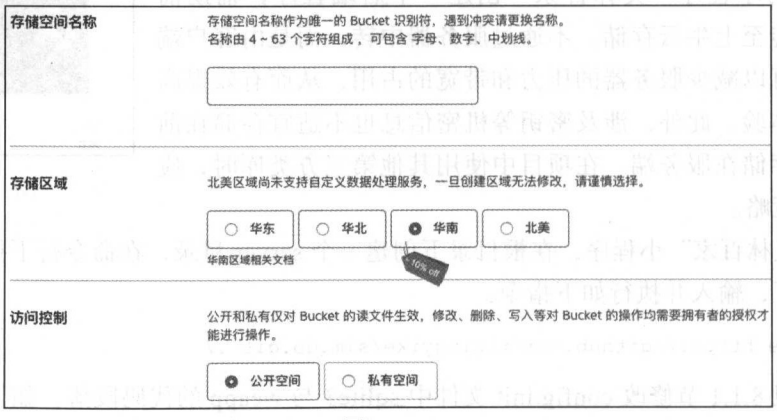


图 10-2

图 10-2 中出现的“存储空间名称”，即配置文件 config.ini 中的 qiniu-scope 字段。

创建存储空间之后，单击右上角的“控制面板”按钮，选择密钥管理，如图 10-3 所示。

图　10-3

在密钥管理页面将看到要找的 AK 和 SK，如图 10-4 所示。

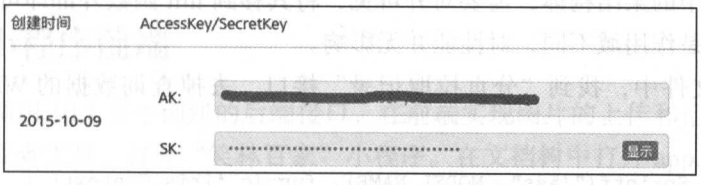


图　10-4

这里的 AK 与 SK 对应于 config.ini 中的 qiniu-access_key 和 qiniu-secret_key，这两个字段是机密信息，只在服务器上存储，要注意保密。

最后，执行 debug.sh 脚本，启动 Web 站点并开启热编译。在浏览器中打开 http://localhost:4000/hi，如果有内容输出，则说明服务端程序启动成功。

10.1.3　Go 语言的作用域

本节将主要介绍 Go 语言的作用域，以及 sim.go 类库是利用作用域的限制提升编码安全性的方法。

首先，打开命令行终端，执行 cd 命令到 server/controller 目录，在 blog.go 的基础上新建 joke.go 文件：

```
cd 笑林百家/server/controller
cp blog.go joke.go
```

然后，打开 joke.go 文件，首先修改结构体的定义，代码如下所示：

```
type (
    Joke struct {
        sim.Model 'xorm:"extends"'
        UserId int64 'json:"user_id"'
        Title       string 'xorm:"char(50)" json:"title"'
        Content     string 'xorm:"text" json:"content"'
        Image       string 'xorm:"tinytext" json:"image"'
    }
    JokePage struct {
        List       []Joke `json:"list"'
        Size       int 'json:"size"'
        Page       int 'json:"page"'    //自1开始计数
        Count      int 'json:"count"'
        TotalPage  int 'json:"total_page"'
    }
)
```

其中，粗体是修改和新增的部分。将结构体的名称修改之后，因为这两个结构体默认是在 init 函数内部定义的，外部不可见，所以下面关于接口实现的代码自然就会出错。在 WebStorm 编辑器中，引用不存在的结构体，代码会高亮显示，在 joke.go 文件中很容易找出出错的名称，可将之替换为新的结构体名称。这是将结构体定义在 init 内部的便利。

如果控制器中的某结构体，需要对外可见，将其移到 init 函数外部即可。定义在 init 函数内部、外部只是作用域不同，对性能并无影响。

在 joke.go 文件中，找到“分页拉取记录”接口，去掉查询数据的 Where 语句，代码如下：

```
sub.Get(fmt.Sprintf("/%ss", MODEL_NAME), func(c *iris.Context) {
    var r sim.Result
    var page,_ = c.URLParamInt("page")
    var size,_ = c.URLParamInt("size")

    if page == 0 {
        page = 1
    }
    if size == 0 {
        size = 1000
    }
    var data = JokePage{Page:page,Size:size}
    var offset = (data.Page - 1) * data.Size
    var my = web.GetWeappUser(c)

    if err := web.DB.Desc("id").Where("user_id = ?", my.ID).Limit(data.Size, offset).
        Find(&data.List); err == nil {
        if total, err := web.DB.Where("user_id = ?", my.ID).Count(new(Joke)); err
            == nil {
            data.Count = int(total)
            data.TotalPage = int(math.Ceil(float64(total) / float64(data.Size)))
        }else{
```

```
            sim.Debug("select count err", err.Error())
        }
        for k,v := range data.List{
            web.DB.ID(v.UserId).Get(&v.User)
            data.List[k].User = v.User
        }
        r.Code = 1
        r.Data = data
    }else{
        sim.Debug("select page err", err.Error())
    }

    c.JSON(200, r)
})
```

其中的粗体是要删除的代码。因为要从 /jokes 接口拉取所有用户上传的笑话，所以不关心用户 ID。

另外，要注意将 MODEL_NAME 修改为 joke，这是容易忽视的地方，修改后代码如下：

```
const MODEL_NAME = "joke"
```

10.2　修改小程序前端

本节将主要调用 10.1 节中创建的后端接口，在前端实现图片的上传和记录保存功能。

使用微信开发者工具，打开“笑林百家”小程序。在文档树中打开 app.js，将内容替换为如下代码：

```
let app = require("./sim.js/index.js")
app.config.enableWeappUserAutoLogin = true
app.config.serverUrlBase = "http://localhost:4000"

App(Object.assign(app, {
    onLaunch() {
        app.getUserInfo()
    }
}))
```

其中的粗体部分是新增的代码。

在以上代码中，serverUrlBase 无论是使用 http://localhost:4000，还是使用 http://localhost:4001，效果是一样的，因为使用 gin 工具代理服务端程序实现热编译时，服务端程序本身使用的是 4001 端口，gin 工具使用的是 4000 端口，两个接口提供的功能是相同的。

参见 server/debug.sh 代码，如下所示：

```
gin -a 4001 -p 4000 run main.go
```

这里的参数 -a 表示 app，指示 main.go 程序使用的端口；参数 -p 表示 port，指示 gin 启用的端口。

10.2.1 使用模板组件实现顶部导航栏

在文档树中选择 pages/index 目录，添加一个 .wxml 文件，如图 10-5 所示。

将该文件命名为 navbar.wxml，打开 pages/index/navbar.wxml 文件，将内容替换为如下标签代码：

```
<view class="weui-navbar">
    <navigator url="index" class="weui-navbar__item {{activeIndex == 0 ? 'weui-
        bar__item_on' : ''}}">
        <view class="weui-navbar__title">笑话 </view>
        </navigator>
        <navigator url="image" class="weui-navbar__item {{activeIndex == 1 ? 'weui-
            bar__item_on' : ''}}">
        <view class="weui-navbar__title">趣图</view>
    </navigator>
</view>
```

以上代码定义一个顶部导航栏，包含两个标签：“笑话”和“趣图”，分别指向相对页面 index 和 image。变量 activeIndex 用于控制当前页面的选中样式，需要在逻辑层中定义。

现在分别打开 pages/index/index.js 和 pages/index/image.js 文件，在页面数据 data 中添加 activeIndex：

```
data: {
    activeIndex:1,
    page: 1,
    picList: []
}
```

其中的粗体为新增代码。

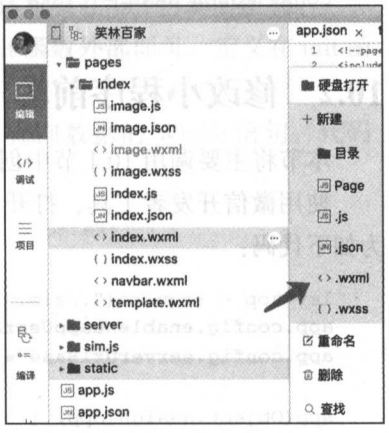


图 10-5

之后分别打开 pages/index/index.wxml 和 pages/index/image.wxml 文件，在顶部添加如下代码：

```
<include src="navbar.wxml"/>
```

以上代码使用 include 关键字引入了 pages/index/navbar.wxml 文件的内容，相当于复制过来使用。

接着，分别在两个文件中的 scroll-view 组件上通过样式添加边距，代码如下：

```
<scroll-view style="height:100%;margin-top:50px" scroll-y bindscrolltolower=
    "loadMore">
    ...
```

```
</scroll-view>
```

其中的粗体为新增代码。完成后的效果如图 10-6 所示。

刷新项目，单击顶部的“导航栏”按钮却没有反应，页面不能跳转。为什么？哪里出错了？

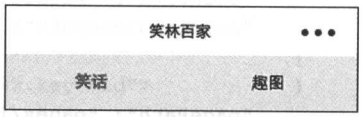


图　10-6

10.2.2　关于 navigator 组件的 open-type 属性

要想解决 10.2.1 节出现的问题，必须先了解 navigator 组件的 open-type 属性。open-type 属性共有 5 个有效值，具体如下。

- navigate：保留当前页面，跳转到应用内的某个页面。
- redirect：关闭当前页面，跳转到应用内的某个页面。
- switchTab：跳转到 tabBar 页面，并关闭其他所有非 tabBar 页面。
- reLaunch：关闭所有页面，打开到应用内的某个页面。
- navigateBack：关闭当前页面，返回上一页面或多级页面。

只有 switchTab、reLaunch 可以跳转到 tabBar 页面。在 10.2.1 节的顶部导航栏中，要切换的页面是 tabBar 页面，只有将 open-type 属性设置为 switchTab 才可以正常工作。关于 navigator 组件的更多信息，参见本书的 14.1 节。

在文档树中打开 pages/index/navbar.wxml 文件，修改 navigator 组件，添加 open-type 属性，代码如下：

```
<navigator url="index" open-type="switchTab" class="weui-navbar__item {{activeIndex
    == 0 ? 'weui-bar__item_on' : ''}}">
    ...
</navigator>
<navigator url="image" open-type="switchTab" class="weui-navbar__item {{activeIndex
    == 1 ? 'weui-bar__item_on' : ''}}">
    ...
</navigator>
```

在 navigator 中使用的 url 地址，如果在 tabBar 中使用了，则必须添加属性 open-type 为 switchTab，否则代码不能正常工作。

10.2.3　在 tabBar 中新增操作按钮

在浏览器中打开网站 http://www.flaticon.com/，参照 5.3 节的方法，分别检索“plus”和“image”，找到合适的图片，保存至 static/image/icon 目录下，如图 10-7 所示。


plus.png


image.png

图　10-7

在文档树中打开 app.json 文件，于左右 tab 中间添加一个新的页面，代码如下：

```
{
    "pagePath": "pages/index/index",
```

```
    "text": "笑谈",
    "iconPath": "/static/image/icon/joke.png",
    "selectedIconPath": "/static/image/icon/joke_on.png"
},
{
    "pagePath": "pages/index/new-record",
    "text": "新增",
    "iconPath": "/static/image/icon/plus.png",
    "selectedIconPath": "/static/image/icon/plus.png"
},
{
    "pagePath": "pages/index/image",
    "text": "趣图",
    "iconPath": "/static/image/icon/funny.png",
    "selectedIconPath": "/static/image/icon/funny_on.png"
}
```

其中的粗体为新增代码。选中状态与非选中状态使用的是同一个图标，看起来更像是按钮。效果如图 10-8 所示。

如果想实现中间的图标比其他略大的效果，可以修改图标，对中间图标采用小边距，其他图标则采用大边距。

图 10-8

10.2.4 使用 icon 组件

使用快捷方法新建 pages/index/new-record 页面，并将其设置为首页。打开 pages/index/new-record.wxml 页面，将内容替换为如下代码：

```
<view class="weui-cells">
    <view class="weui-cell">
        <view class="weui-cell__bd">
            <textarea bindinput="onInputJokeText" class="weui-textarea" placeholder=
                "请输入文本" maxlength="200" style="height: 3.3em" />
            <view class="weui-textarea-counter">200</view>
        </view>
    </view>
</view>
<view class="weui-btn-area" style="margin-top:20rpx">
    <view class="weui-flex">
        <view class="weui-flex__item v-center">
            <image bindtap="uploadImage" src="/static/image/icon/image.png" style=
                "width:60rpx;height:60rpx;margin-right:10rpx"></image>
                <view wx:if="{{newJoke.image}}"><icon type="success_no_circle"
size="16"></icon>已上传</view>
        </view>
        <view class="weui-flex__item" style="text-align:right;line-height:0;">
            <button bindtap="save" class="weui-btn mini-btn" type="primary"
                size="mini">保存</button>
```

```
        </view>
    </view>
</view>
<!-- 自建笑话列表 -->
<view class="weui-panel" wx:for="{{jokePage.list}}" wx:key="id">
    <view class="weui-panel__bd">
        <view class="weui-media-box weui-media-box_text">
            <text>{{item.content}}</text>
            <view wx:if="{{item.image}}" class="weui-media-box__desc">
                <image  src="{{item.image}}"  style="width:100%"  mode="aspectFit"
                    bindtap="preview" data-url="{{item.image}}" />
            </view>
        <view class="weui-media-box__info">
            <view  class="weui-media-box__info__meta">作者: {{item.user.nickName}}
                </view>
            </view>
        </view>
    </view>
</view>
```

使用文本域组件 textarea 输入文本。单击“图片”图标上传图像之后，会显示“已上传”的提示，如图 10-9 所示。

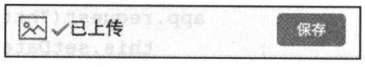


图　10-9

绑定的函数 uploadImage 用于上传图片，函数 save 用于保存生成的新笑话。“笑话”新建区域下方是笑话列表，绑定的是 jokePage 对象。

10.2.5　在小程序中直接上传图片

在文档树中打开 pages/index/new-record.js 文件，修改代码如下：

```
let app = getApp()
Page({
    data: {
        newJoke:{
            content:"",
            image:""
        },
        jokePage:{}
    },
    save(){
        let joke = this.data.newJoke
        console.log(joke)
        if (joke.content || joke.image){
            app.request("http://localhost:4000/joke",joke,{method:"POST"}).
                then(res => {
                console.log(res)
                this.setData({
                    newJoke:{}
                })
```

```
                this.retrieve()
            })
        }else{
            wx.showModal({
                title: '提示',
                content: '未输入文本或上传图片',
            })
        }
    },
    onInputJokeText(e){
        this.data.newJoke.content = e.detail.value
    },
    uploadImage(){
        app.selectAndUploadImageToQiniu().then(res =>{
            this.setData({
                "newJoke.image":res
            })
            console.log(res)
        })
    },
    retrieve(){
        app.request("http://localhost:4000/jokes").then(res => {
            this.setData({
                jokePage: res.data
            })
        })
    },
    onLoad (options) {
        this.retrieve()
    },
    preview(e) {
        var urls = [e.target.dataset.url]
        wx.previewImage({
            urls: urls
        })
    }
})
```

在 save 函数中，使用了小程序的 wx.showModal 接口，用于提示用户：

```
wx.showModal({
    title: '提示',
    content: '未输入文本或上传图片',
})
```

title 是提示窗口的标题，content 是提示内容。效果如图 10-10 所示。

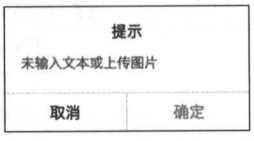


图 10-10

在函数 uploadImage 中，使用了 sim.js 类库中的新方法 selectAndUploadImageToQiniu：

```
app.selectAndUploadImageToQiniu().then(res =>{
    this.setData({
        "newJoke.image":res
    })
    console.log(res)
})
```

该方法会自动选择一张图片并上传至七牛云存储。

不需要额外设置其他内容，因为我们已经在 10.1.1 节的 config.ini 配置中开启了云上传支持。打开 sim.js/index.js 文件，找到 uploadToQiniu 函数，从这个函数中可以看出，其图片的上传是直接向七牛云存储的服务器上传，仅是从自己的服务器获取 uptoken，而 uptoken 是标识七牛云账号的信息。

在用户手机端直接向七牛的服务器上传，避免从自己的服务器中转，可以有效节省流量，减轻服务器压力。在用户增多以后，由于七牛使用的是云主机，用户就近上传，还避免了扩展服务器功能的麻烦。

app. selectAndUploadImageToQiniu 函数使用了七牛的域名 https://up.qbox.me，如果提交版本审核，记得把该域名添加到服务器域名列表中。

10.3　下载源码

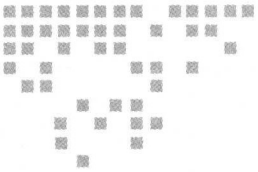
在本章的学习过程中，如果遇到问题，可以通过如下方式来解决。

- 加入小程序微信群与作者及其他读者一同探讨。微信扫描前言中的二维码，关注“艺述思维”，发送“小程序”进群。
- 下载作者的源码，对照查找问题。在公众号中发送“笑林百家 10.2”获取下载地址。服务端的源码在小程序项目的 server 目录中。

第三篇 Part 3

实用组件篇

- 第 11 章　容器组件
- 第 12 章　基础内容组件
- 第 13 章　表单组件
- 第 14 章　多媒体及其他组件



Chapter 11 第 11 章

容器组件

读者可以在微信公号“艺述思维”中回复“微信小程序从 0 到 1 练习”，下载随章示例程序。本书第 11～14 章的练习代码在这个示例中都能找到。该示例项目是基于微信官方的小程序示例修改的。

容器组件用于装载其他组件，目前小程序的容器组件有 view、scroll-view、swiper、movable-view、cover-view 等。

11.1 view

view 是小程序 UI 设计中的万能视图，能独立表现文本，也能装载其他组件，很多看似复杂的 UI 效果都是基于最普遍的 view 实现的。

view 具有以下属性。

- hover-class：指定按下去的样式类。当 hover-class="none" 时，表示没有点击态效果。使用 sim.js 中的“view-hover-class”，有一个 80% 的高亮效果。默认为“none”。
- hover-start-time：按住后多久出现点击态，单位为 ms。
- hover-stay-time：手指松开后点击态保留的时间，单位为 ms。

在 style 属性中设置 flex-direction 样式为 row，可以实现视图块的横向布局，这也是子组件默认的布局，代码如下所示：

```
<view class="flex-wrp" style="flex-direction:row;">
    <view hover-class="view-hover-class" hover-start-time="100" hover-stay-
        time="200" class="flex-item demo-text-1"></view>
    <view hover-class="view-hover-class" class="flex-item demo-text-2"></view>
    <view hover-class="view-hover-class" class="flex-item demo-text-3"></view>
```

```
</view>
```

上述代码将 hover-stay-time 属性设置为 200ms，这也是人类易于感知的最小时间，效果如图 11-1 所示。

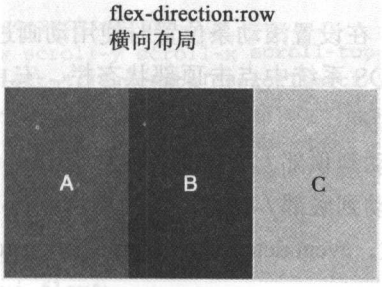


图　11-1

在 style 属性中设置 flex-direction 样式为 column，可以实现子组件的竖向布局，代码如下所示：

```
<view class="flex-wrp" style="flex-direction:column;">
    <view hover-class="view-hover-class" class="flex-item
        flex-item-V demo-text-1"></view>
    <view hover-class="view-hover-class" class="flex-item
        flex-item-V demo-text-2"></view>
    <view hover-class="view-hover-class" class="flex-item
        flex-item-V demo-text-3"></view>
</view>
```

hover-class 属性设置长按样式为 view-hover-class，就会有一个高度效果，如图 11-2 所示。

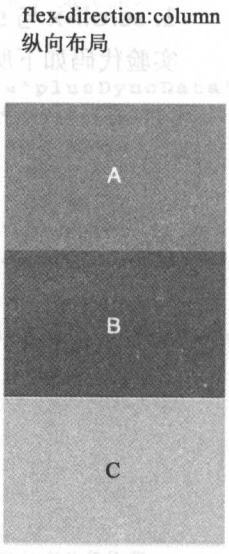


图　11-2

11.2　scroll-view

scroll-view 是赋予子组件滚动功能的容器，在第 2 章“豆豆电影”小程序的列表页中已经使用过。使用竖向滚动时，需要给 <scroll-view/> 一个固定高度，通过 WXSS 设置 height。

scroll-view 具有如下属性。

- scroll-x：允许横向滚动。
- scroll-y：允许纵向滚动。
- upper-threshold：距顶部 / 左边多远时（单位 px），触发 scrolltoupper 事件，默认为 50px。
- lower-threshold：距底部 / 右边多远时（单位 px），触发 scrolltolower 事件，默认为 50px。

- scroll-top：设置竖向滚动条位置。
- scroll-left：设置横向滚动条位置。
- scroll-into-view ：值应为某子元素的 id。若设置哪个方向可滚动，则在哪个方向滚动到该元素。
- scroll-with-animation：在设置滚动条位置时使用动画过渡。
- enable-back-to-top ：iOS 系统中点击顶部状态栏、安卓系统中双击标题栏时，滚动条返回顶部，只支持竖向。
- bindscrolltoupper：滚动到顶部 / 左边，会触发 scrolltoupper 事件。
- bindscrolltolower：滚动到底部 / 右边，会触发 scrolltolower 事件。
- bindscroll：滚动时触发，event.detail 等于 {scrollLeft, scrollTop, scrollHeight, scrollWidth, deltaX, deltaY}。

对于上述属性，可能有读者会有如下疑惑。

1. scroll-x 与 scroll-y 能否同时设置为 true，实现四面八方可同时滚动

实验代码如下所示：

```
<scroll-view scroll-y scroll-x style="height: 300rpx;">
    <view id="demo1" style="width:1000rpx" class="scroll-view-item demo-
        text-1"></view>
    <view id="demo2" class="scroll-view-item demo-text-2"></view>
    <view id="demo3" class="scroll-view-item demo-text-3"></view>
</scroll-view>
```

答案是可以的，效果如图 11-3 所示。

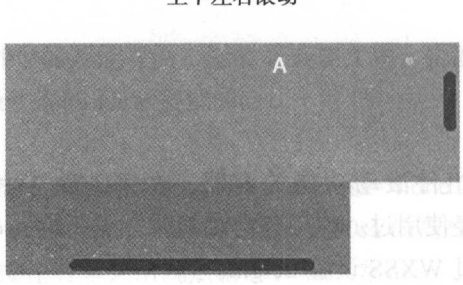


图 11-3

关键是必须让子组件的宽、高均大于 scroll-view 容器，设置第一个子组件 width 等于 1000rpx 是必须要有的代码。

注意，布尔值属性如 scroll-y、scroll-x 等，不能写属性值，直接写个属性就可以了。

2. 如何手动设置竖向滚动条位置

首先，将 scroll-top 或 scroll-left 属性绑定到一个 data 变量上，然后动态改变变量的值。

代码如下：

```
<view class="page-section">
    <view class="page-section-title center">
            <text>手动设置竖向滚动条位置</text>
        </view>
        <view class="page-section-spacing">
            <scroll-view scroll-y scroll-x scroll-top="{{dyncData['scrollTop1']}}"
                scroll-left="{{dyncData['scrollLeft1']}}" style="height: 300rpx;">
            <view style="width:1000rpx" class="scroll-view-item demo-text-1"></view>
            <view class="scroll-view-item demo-text-2"></view>
            <view class="scroll-view-item demo-text-3"></view>
        </scroll-view>
    </view>
    <view class="weui-btn-area">
        <view class="weui-flex">
            <view class="weui-flex__item center">
                <button data-name="scrollTop1" bindtap="plusDyncData"
                    class="weui-btn mini-btn" size="mini">横向自动</button>
            </view>
            <view class="weui-flex__item  center">
                <button data-name="scrollLeft1" bindtap="plusDyncData"
                    class="weui-btn mini-btn" size="mini">竖向自动</button>
            </view>
        </view>
    </view>
</view>
```

其中的粗体部分是主要代码。

接着通过两个按钮控制两个变量自增，实现 scroll-top、scroll-left 的动态绑定。plusDyncData 函数的代码如下所示：

```
plusDyncData(e){
    var filedName = e.currentTarget.dataset.name
    var dyncData = this.data.dyncData
    dyncData[filedName] = (dyncData[filedName] || 0)+5
    this.setData({
        dyncData:dyncData
    })
}
```

为了避免重复声明变量和函数，笔者声明了一个万能的 dyncData 对象，将变量名称放在了 dataset 中动态设置。这样就可以实现想要的效果，实现的效果如图 11-4 所示。

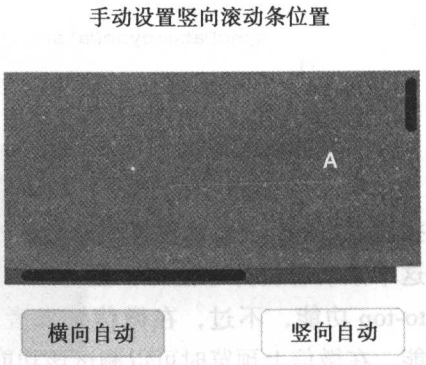


图　11-4

3. 如何实现动态滚动到某子组件

实践代码如下：

```
<view class="page-section">
    <view class="page-section-title">
        <text>滚动到视图</text>
    </view>
    <view class="page-section-spacing">
        <scroll-view scroll-y enable-back-to-top scroll-with-animation scroll-into-
            view="{{dyncData['scrollIntoView1']}}" style="height: 300rpx;">
            <view class="scroll-view-item demo-text-1"></view>
            <view class="scroll-view-item demo-text-1"></view>
            <view class="scroll-view-item demo-text-1"></view>
            <view class="scroll-view-item demo-text-1"></view>
            <view id="scrollIntoView1" class="scroll-view-item demo-text-2"></view>
            <view class="scroll-view-item demo-text-3"></view>
        </scroll-view>
    </view>
    <view class="weui-btn-area">
        <view class="weui-flex">
            <view class="weui-flex__item center">
                <button data-name="scrollIntoView1" data-value="scrollIntoView1"
                    bindtap="setDyncData" class="weui-btn mini-btn" size="mini">
                    一下滚动到倒数第二个子组件</button>
            </view>
        </view>
    </view>
</view>
```

其中的粗体部分为主要代码。scroll-with-animation 属性用于启用滚动动画。enable-back-to-top 属性则用于启用双击标题栏返回顶部。为便于复制复用，笔者声明了一个万能的 setDyncData 函数，通过组件的 dataset 来设置 name、value，代码如下：

```
setDyncData(e){
    var filedName = e.currentTarget.dataset.name
    var filedValue = e.currentTarget.dataset.value
    var dyncData = this.data.dyncData
    dyncData[filedName] = filedValue
    this.setData({
        dyncData:dyncData
    })
}
```

实现的效果如图 11-5 所示。

单击图 11-5 中的“一下滚动到倒数第二个子组件”按钮，scroll-view 视图将滚动到 B 视图，带有动画。虽然这个滚动容器并非全屏，但仍然可有效开启 enable-back-to-top 功能，不过，在微信开发者工具中不能测试该功能，在微信上预览时可以测试该功能。

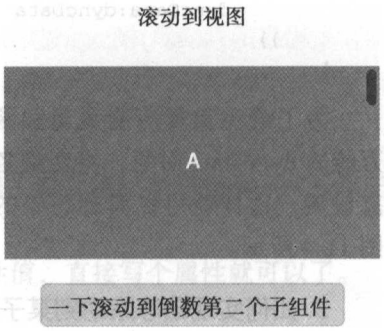


图 11-5

4. 如何监听页面滚动到底部、顶部

代码如下：

```
<view class="page-section">
    <view class="page-section-title center">
        <text>Vertical Scroll\n纵向滚动，带底、顶部滚动测试</text>
    </view>
    <view class="page-section-spacing">
        <scroll-view scroll-y style="height: 300rpx;" bindscrolltoupper="scrollT
        oSide" bindscrolltolower="scrollToSide" bindscroll="scroll">
            <view id="demo1" class="scroll-view-item demo-text-1"></view>
            <view id="demo2" class="scroll-view-item demo-text-2"></view>
            <view id="demo3" class="scroll-view-item demo-text-3"></view>
        </scroll-view>
    </view>
</view>
```

其中的粗体部分为主要代码。无论是滚动到底部，还是顶部，都是将事件绑定到同一个函数：

```
scrollToSide(e) {
    console.log(e)
    if (e.detail.direction == "top"){
        wx.showToast({
            title: '滚动到达顶部',
        })
    } else if (e.detail.direction == "bottom"){
        wx.showToast({
            title: '滚动到达底部',
        })
    }
}
```

通过 event.detail.direction 可检测方向，有效值包括 top、bottom、left、right 等，具体值依据滚动方向而不同。这一点在官方文档中并无说明，在使用时须考虑可能的变化。

运行效果如图 11-6 所示。

图　11-6

单次滚动到边界的事件不会频繁触发，仅会触发一次。

在 scroll 函数中，仅会打开事件对象，代码如下：

```
scroll: function(e) {
    console.log(e)
}
```

event.detail 包含了事件的全部信息，如图 11-7 所示。

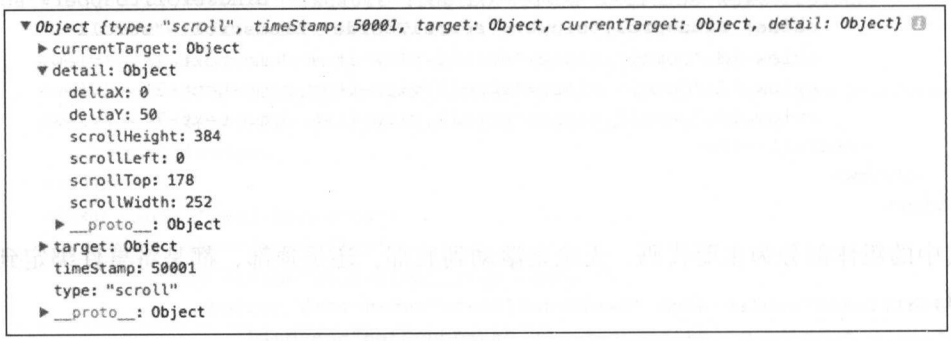


图 11-7

其中 deltaY 代表此次竖向滚动的单次步距，deltaX 同理。

需要说明的是，scroll-view 视图的滚动步距目前尚不能自定义，在 scroll-view 组件上设置 deltaY 属性无效，但也不会报错。小程序组件具有 js 对象的优点，可以随意动态扩展。代码如下：

```
<scroll-view deltaY="10" scroll-y style="height: 300rpx;" bindscrolltoupper="scr
    ollToSide" bindscrolltolower="scrollToSide" bindscroll="scroll">
...
</scroll-view>
```

11.3 swiper

swiper 是滑动视图容器，第 2 章中曾使用它实现过 splash 功能。

swiper 组件主要具有以下属性。

- indicator-dots：是否显示面板指示点。
- indicator-color：指示点颜色，默认为 0.3，即透明的白色。
- indicator-active-color：当前选中的指示点颜色，默认为白色。
- autoplay：是否自动切换。
- current：当前所在的页面，自 0 计数。
- interval：自动切换时间间隔，单位 ms。
- duration：滑动动画时长。

- circular：是否采用衔接滑动。
- vertical：滑动方向是否为纵向，默认为 false。
- bindchange ：current 改变时会触发 change 事件，其中 event.detail = {current: current, source: source}。

swiper 组件虽然属性较多，但是并不复杂。小程序组件的属性，存在很多相通之处，一通则百通。

对应上述属性的实践代码如下：

```
<view class="page-section page-section-spacing swiper">
    <swiper bindchange="traceEvent" circular="{{circular}}" indicator-dots="{{indicatorDots}}"
      auto play="{{autoplay}}" interval="{{interval}}" duration="{{duration}}">
        <block wx:for="{{background}}" wx:key="*this">
            <swiper-item>
                <view class="swiper-item {{item}}"></view>
            </swiper-item>
        </block>
    </swiper>
</view>
<view class="page-section" style="margin-top: 40rpx;margin-bottom: 0;">
    <view class="weui-cells weui-cells_after-title">
        <view class="weui-cell weui-cell_switch">
            <view class="weui-cell__bd">指示点</view>
            <view class="weui-cell__ft">
                <switch data-name="indicatorDots" checked="{{indicatorDots}}"
                    bindchange="onSwitchChange" />
            </view>
        </view>
        <view class="weui-cell weui-cell_switch">
            <view class="weui-cell__bd">自动播放</view>
            <view class="weui-cell__ft">
                <switch data-name="autoplay" checked="{{autoplay}}" bindchange=
                    "onSwitchChange" />
            </view>
        </view>
        <view class="weui-cell weui-cell_switch">
            <view class="weui-cell__bd">采用衔接滑动</view>
            <view class="weui-cell__ft">
                <switch data-name="circular" checked="{{circular}}" bindchange=
                    "onSwitchChange" />
            </view>
        </view>
    </view>
</view>
<view class="page-section page-section-spacing">
    <view class="page-section-title">
        <text>幻灯片切换时长(ms)</text>
        <text class="info">{{duration}}</text>
    </view>
```

```
    <slider data-name="duration" bindchange="onSliderChange" value="{{duration}}"
        min="500" max="2000" />
    <view class="page-section-title">
        <text>自动播放间隔时长(ms)</text>
        <text class="info">{{interval}}</text>
    </view>
    <slider data-name="interval" bindchange="onSliderChange" value="{{interval}}"
        min="2000" max="10000" />
</view>
```

其中的粗体部分是关键代码。这个 demo 包括了 swiper 组件的所有属性。

swiper-item 是 swiper 的子组件，宽高自动设置为 100%。swiper-item 是一种特殊的容器组件。有几个 swiper-item，就代表能翻几页。swiper-item 内部可放置任何可视内容。

为了简化编写，三个布尔值属性 autoplay、circular、vertical 可使用可统一的事件函数 onSwitchChange，代码如下：

```
onSwitchChange(e){
    var fieldName = e.currentTarget.dataset.name
    var data = {}
    data[fieldName] = !(data[fieldName] || false)
    this.setData(data)
}
```

依据每个组件都有的自定义对象 dataset，向逻辑层传递需要改变的布尔值名称，省去了为每个 switch 组件书写一个函数的麻烦。给两个 slider 组件使用的 onSliderChange 函数与之类似，代码如下：

```
onSliderChange(e){
    var fieldName = e.currentTarget.dataset.name
    var data = {}
    data[fieldName] = e.detail.value
    this.setData(data)
}
```

onSwitchChange 函数的实现方法与 onSliderChange 类似，都是先通过 dataset 区别变量名，然后动态创建一个 data 对象，并调用 setData 强制渲染的。直接使用 setData({fieldName :xx}) 是不可以的，JS 会将 fieldName 视为键名，而不是它的值，此处也不可以使用模板字符串，代码如下：

```
onSliderChange(e){
    var fieldName = e.currentTarget.dataset.name
    this.setData({`${fieldName}`:e.detail.value})
}
```

完成后的效果如图 11-8 所示。

图 11-8

关于衔接滑动，是指滑到最后一片时，若继续滑动，则是第一

张。如果不设置 circular 为 true，那么滑动到最后一片时，则只能回头向前滑动。自动播放也会先滑到开头再播放。

在 change 的事件函数中，event.detail.source 代表滑动动画变更的原因，其具有两个有效值，具体如下。

- autoplay：是自动播放导致 swiper 变化。
- touch：是用户滑动引起 swiper 变化。

本节不仅练习了 swiper 组件，还练习了 switch 和 slider 组件。

11.4 movable-view

movable-view 是一个可移动的视图容器，与 movable-area 搭配使用，在 movable-area 内可以拖曳滑动。对于 movable-area，必须设置 width 和 height 属性，若不设置则默认为 10px×10px，在这么小的区域内任何组件几乎都无法拖动。

movable-view 主要具有以下属性。

- direction：movable-view 的移动方向，属性值有 all、vertical、horizontal、none 等。
- inertia：movable-view 是否带有惯性，默认为 false。
- out-of-bounds：超过可移动区域之后，movable-view 是否还可以移动，默认为 false。
- x：定义 x 轴方向的偏移，如果 x 的值不在可移动的范围之内，那么其会自动移动到可移动范围。改变 x 的值会触发动画。
- y：定义 y 轴方向的偏移，如果 y 的值不在可移动的范围之内，那么其会自动移动到可移动范围。改变 y 的值会触发动画。
- damping：阻尼系数，用于控制 x 或 y 改变时的动画和过界回弹的动画，值越大移动越快。
- friction：摩擦系数，用于控制惯性滑动的动画，值越大摩擦力越大，滑动也会越快停止；其值必须大于 0，否则会被设置成默认值。

主要实践代码如下：

```
<view class="page-section">
    <view class="page-section-spacing center">
        <movable-area style="height: 200px;width: 200px;background: red;">
            <movable-view damping="{{damping}}" friction="{{friction}}"
                inertia="{{inertia}}" out-of-bounds="{{outOfBounds}}" x="{{pos.x}}"
                y="{{pos.y}}" style="height: 50px; width: 50px; background: blue;"
                direction="{{direction}}">
            </movable-view>
        </movable-area>
    </view>
    <view class="weui-btn-area">
```

```
            <view class="weui-flex">
                <view class="weui-flex__item center">
                <button data-name="pos" data-value="{{target}}" bindtap="setNameValue"
                    class="weui-btn mini-btn" size="mini">单击移至(30px, 30px)</button>
                </view>
            </view>
        </view>
    </view>
    <view class="page-section" style="margin-top: 40rpx">
        <view class="weui-cells weui-cells_after-title">
            <view class="weui-cell weui-cell_switch">
                <view class="weui-cell__bd">使用惯性</view>
                <view class="weui-cell__ft">
                    <switch data-name="inertia" checked="{{inertia}}" bindchange=
                        "onSwitchChange" />
                </view>
            </view>
            <view class="weui-cell weui-cell_switch">
                <view class="weui-cell__bd">允许越界</view>
                <view class="weui-cell__ft">
                    <switch data-name="outOfBounds" checked="{{outOfBounds}}" bindchange=
                        "onSwitchChange" />
                </view>
            </view>
        </view>
    </view>
    <view class="page-section">
        <view class="page-section-title">
            <text>阻尼系数</text>
            <text class="info">{{damping}}</text>
        </view>
        <slider data-name="damping" bindchange="onSliderChange" value="{{damping}}"
            show-value min="10" max="100" />
        <view class="page-section-title">
            <text>摩擦系数</text>
            <text class="info">{{friction}}</text>
        </view>
        <slider data-name="friction" bindchange="onSliderChange" value="{{friction}}"
            show-value min="1" max="10" />
        <view class="page-section-title">
            <text>滑动方向</text>
            <text class="info">{{direction}}</text>
        </view>
        <radio-group data-name="direction" class="radio-group" bindchange=
            "onRadioChange">
            <label class="radio" wx:for="{{['all', 'vertical','horizontal','none']}}"
                wx:key="*this">
                <radio value="{{item}}" checked="{{item == direction}}"/>{{item}}
            </label>
        </radio-group>
    </view>
```

那么，如何在列表渲染中直接绑定字面数组呢？

如果把一个静态数组写进页面初始数据对象 data 中，只使用一次则有些浪费。而如果多次输入重复的代码又会显得很枯燥，这时候可以使用字面数组直接绑定于列表渲染中，示例代码如下：

```
<radio-group data-name="direction" class="radio-group" bindchange="onRadioChange">
    <label class="radio" wx:for="{{['all', 'vertical','horizontal','none']}}"
        wx:key="*this">
        <radio value="{{item}}" checked="{{item == direction}}"/>{{item}}
    </label>
</radio-group>
```

其中的粗体部分就是字面数组，如果不以中括号“[]”括起来，那么循环的将是字符串，而非数组。

radio-group 是单项选择器，内部由多个 <radio/> 组成。

上面的视图层代码对应的逻辑层代码如下：

```
let app = getApp()

Page(Object.assign(app.page, {
    data: {
        outOfBounds:true,
        inertia:true,
        friction:2,
        damping:20,
        direction: "all",
        target:{x:30,y:30}
    }
}))
```

以上代码非常简洁，在视图层使用的 onSwitchChange、onSliderChange、onRadioChange 函数并没有定义在这里。那么，它们被定义在哪里了？

打开 sim.js/lib/page.js 文件，即可看到这些函数的定义。由于它们具有稳定、重复的逻辑结构，所以将它们抽象到 sim.js 类库中，以便后续重用。

函数 Object.assign 会将当前文件的页面对象（见上面代码中的粗体部分）复制到 app.page 对象中，以覆盖式并集的方式组成新的页面对象，页面对象中定义的函数可以覆盖掉 app.page 中预定义的函数。

关于属性 direction，截至作者发稿时尚不支持动态绑定。切换 radio 选项，对移动方向无影响。

完成后的效果如图 11-9 所示。

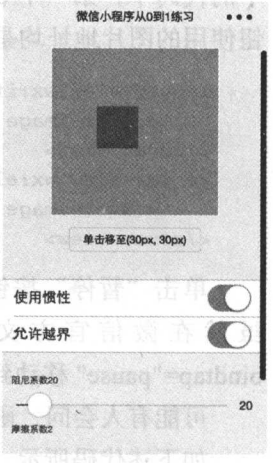


图　11-9

11.5 cover-view

cover-view 是一个特殊的视频容器，覆盖在原生组件之上，可覆盖的原生组件包括 map、video、canvas，支持嵌套，支持内嵌 cover-image 组件。cover-image 组件与 image 类似，但其仅有一个 src 属性。

使用 cover-view、cover-image，可以在原生的 video 组件上实现一个自定义的播放控件。它们已经支持实现的功能包括播放 / 暂停、全屏 / 退出全屏、显示播放进度 / 拖动播放、设置播放倍速等。

下面将使用 cover-view 组件实现播放、暂停的功能控件。

如图 11-10 所示，单击“播放”按钮，视频开始播放。图 11-11 所示的是单击“暂停”按钮，视频暂停的画面截图。

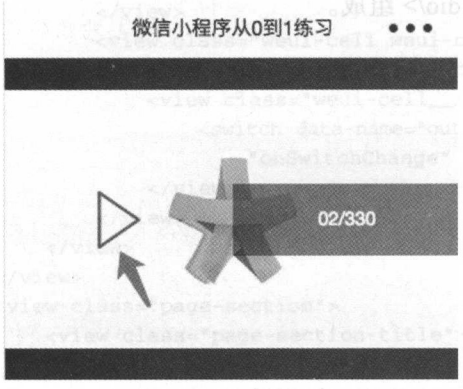


图 11-10

图 11-11

其中，“播放”和“暂停”按钮是嵌套于 cover-view 容器内的 cover-image 组件。在如下的代码中，第一个 cover-image 是“暂停”按钮，第二个是“播放”按钮。这两个图像按钮使用的图片地址均是相对地址，与当前页面位于同一目录下。

```
<cover-view wx:if="{{playing}}" class="pause" bindtap="pause">
    <cover-image class="img" src="pause-button.png" />
</cover-view>
<cover-view wx:else class="play" bindtap="play">
    <cover-image class="img" src="play-button.png" />
</cover-view>
```

单击“暂停”按钮将会触发绑定在其父容器 cover-view 上的 tap 事件函数“ pause”。虽然在微信官方文档中，cover-image 仅声明了一个 src 属性。但是如果将代码 bindtap="pause" 移动到子组件 cover-image 之上，那么其一样可以工作。

可能有人会问，函数 pause 是如何工作的？

如下述代码所示，在逻辑层代码中，首先在页面周期函数 onReady 中，使用 wx.create-VideoContext 接口创建一个 VideoContent 对象。该接口仅接收一个 id 参数，即视图层视频

组件 video 的 id：

```
onReady() {
    this.videoCtx = wx.createVideoContext('myVideo')
}
```

VideoContent 对象是一个控制视频源播放、展示等行为的对象。它有多个方法，参见本书 14.4 节的 video 组件。

然后，在 pause、play 方法中，就可以通过 this.videoCtx 控制视图层的视频组件了。如下代码即为视图层绑定的 pause 函数，它直接调用了 videoCtx 对象的 pause 方法：

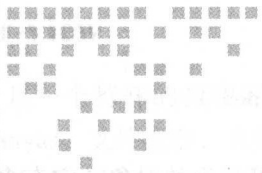
```
pause() {
    this.videoCtx.pause()
}
```

play 方法的实现与 pause 类似。两个按钮隐藏 / 显示的切换是通过页面变量 playing 来控制的。在如下代码中，会通过对 play、pause、ended 事件的绑定，监听其播放状态：

```
<video id="myVideo" autoplay="{{false}}" bindplay="onPlay" bindpause="onPause"
    bindended="onPause" bindtimeupdate="onTimeUpdate" src=".."
    controls="{{false}}" event-model="bubble" style="width:100%">
    ...
</video>
```

onPause 与 onPlay 方法是通过页面方法 setData 切换页面变量 playing 的。示例代码如下：

```
onPause() {
    this.setData({
        playing: false
    })
},
onPlay(){
    this.setData({
        playing: true
    })
}
```



Chapter 12 第 12 章

基础内容组件

基础内容组件包括 icon、text、rich-text、progress，分别用于实现图标、普通文本、富文本、进度条的显示。

12.1 icon

1. 12 个 icon 图标

在微信官方文档中，登记在册的 icon 图标有 9 个：

```
success, success_no_circle, info, warn, waiting, cancel, download, search, clear
```

除去这 9 个之外，还有 3 个图标可以使用，但并未公布在官网文档中。观察一下上面的图标类型，不难发现，“_no_circle”加在“success”后面构成了新类型“success_no_circle”。有“_no_circle”就有“_circle”，将其加在“success”后面就是“success_circle”。“success_circle”类型虽然暂未见公开，但可以使用。下面将通过代码实践来证实这个猜测。

下面将 8 个基本图标类型 success、info、warn、waiting、cancel、download、search、clear，分别与“_no_circle”及“_circle”组合，组成了 24 个类型，代码如下：

```
onLoad(){
    const ORI_TYPES = ["success", "info", "warn", "waiting", "cancel",
        "download", "search", "clear"]
    let arr = []
    for (var k in ORI_TYPES){
        let iconType = ORI_TYPES[k]
        arr.push(iconType, iconType + "_no_circle", iconType +"_circle")
```

```
    }
    console.log(arr)
    this.setData({
        iconTypes2: arr
    })
}
```

将这 24 个类型使用 wx:for 渲染出来，代码如下：

```
<view class="weui-cells weui-cells_after-title">
    <view wx:for="{{iconTypes2}}" class="weui-cell">
        <view class="weui-cell__hd" style="width:60rpx">
            <icon type="{{item}}" size="20" />
        </view>
        <view class="weui-cell__bd">{{item}}</view>
    </view>
</view>
```

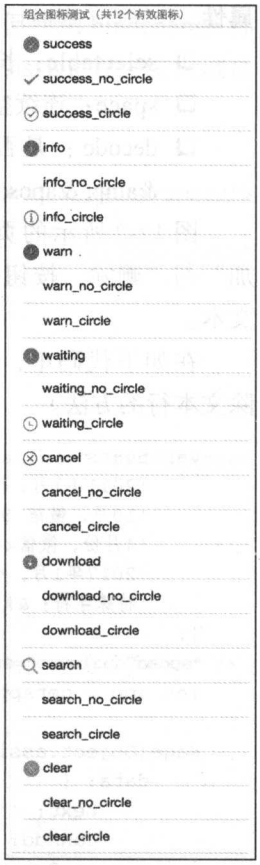


图　12-1

图 12-1 是渲染效果的截图，其中共有 12 个有效的图标。比官方多出来的三个图标分别是 success_circle、info_circle 和 waiting_circle。

2. 如何使用 icon 图标

icon 图标的使用非常简单，它只有三个属性，具体如下。

- ❑ type：icon 的类型，有效值如图 12-1 所示。
- ❑ size：icon 的大小，单位为 px，默认为 23。
- ❑ color：icon 的颜色，同 css 的 color，默认为图标自身的设计色，与图标类型有关。

如下代码是 icon 图标的使用示例：

```
<icon type="success" size="23" color="yellow" />
```

其中，color 属性接受 4 种形式的值，具体如下。

- ❑ RGB 颜色：如“rgb(255, 0, 0)”。
- ❑ RGBA 颜色：如“rgba(255, 0, 0, 0.3)”。
- ❑ 16 进制颜色：如“#FF0000”。
- ❑ 预定义的颜色：如“red”。

预定义的颜色有 100 多种，仅记住常用的一些就可以了，如 red、orange、yellow、green、blue、purple，其他颜色适合先在设计软件中调好色值，再复制过来使用。

12.2　text

本节将主要介绍基础内容组件 text 的使用，text 组件用于展示简易文本，其具有以下

属性。

- selectable：长按下文本是否可选，默认为 false。
- space：连续空格显示策略，默认不启用，有效性有 ensp、emsp、nbsp。
- decode：是否解码，可以解析的有“   < > & '   ”。

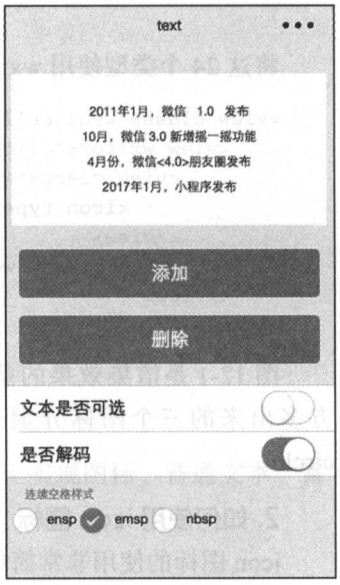


图 12-2

图 12-2 所示的页面用于测试 text 组件的三个属性。“添加”与“删除”按钮分别用于显示新增一行文本与减少一行文本。

在如下代码中，add 是添加文本行的方法，remove 是删除文本行的方法。

```
var texts = [
    '2011年1月，微信 1.0 发布',
    '10月，微信 3.0 新增摇一摇功能',
    '4月份，微信<4.0>朋友圈发布',
    '2017年1月，小程序发布',
    '特殊字符：&|''|?|?结束',
];

let app = getApp()

Page(Object.assign(app.page, {
    data: {
        text: '',
        canAdd: true,
        canRemove: false
    },
    extraLine: [],
    add(e) {
        this.extraLine.push(texts[this.extraLine.length])
        this.setData({
            text: this.extraLine.join('\n'),
            canAdd: this.extraLine.length < texts.length,
            canRemove: this.extraLine.length > 0
        })
    },
    remove(e) {
        if (this.extraLine.length > 0) {
            this.extraLine.pop()
            this.setData({
                text: this.extraLine.join('\n'),
                canAdd: this.extraLine.length < texts.length,
                canRemove: this.extraLine.length > 0,
            })
        }
    }
}))
```

在如下代码中，粗体部分通过 bindtap 绑定了 add、remove 方法。但使用 bindchange 绑定的 onSwitchChange 方法并不存在于这段代码之中，它是通过 Object.assign 方法注入该页面的，源码见开源类库：https://github.com/rixingyike/sim.js。

```
<view class="weui-btn-area">
    <button disabled="{{!canAdd}}" bindtap="add" class="weui-btn" type="primary">
        添加</button>
    <button disabled="{{!canRemove}}" bindtap="remove" class="weui-btn"
        type="primary">删除</button>
</view>
<view class="weui-cells">
    <view class="weui-cell weui-cell_switch">
        <view class="weui-cell__bd">文本是否可选</view>
        <view class="weui-cell__ft">
            <switch data-name="selectable" checked="{{selectable}}" bindchange=
                "onSwitchChange" />
        </view>
    </view>
    <view class="weui-cell weui-cell_switch">
        <view class="weui-cell__bd">是否解码</view>
        <view class="weui-cell__ft">
            <switch data-name="decode" checked="{{decode}}" bindchange="onSwitchChange" />
        </view>
    </view>
</view>
```

12.3　rich-text

rich-text 是微信小程序的富文本组件，可以渲染部分 html 标签，全局支持 class 和 style 属性，但不支持 id 属性。rich-text 弥补了 text 组件在文本渲染上的不足。如下代码是 rich-text 组件的声明方式：

```
<rich-text nodes="{{nodes}}"></rich-text>
```

rich-text 组件支持默认事件，包括 tap、touchstart、touchmove、touchcancel、touchend 和 longtap。

1. 以数组方式定义 rich-text 的 nodes

rich-text 组件在渲染时支持两种绑定数据的方式，第一种是使用结构化的数组，第二种是直接使用 html 字符串。如下面的代码所示，省略号上方的是逻辑层代码，采用的是 json 结构；省略号下方的是标签代码，直接将 nodes 数组绑定于 rich-text 渲染：

```
nodes1: [{
    name: 'div',
    attrs: {
        class: 'div_class',
```

```
        style: 'line-height: 60px; color: red;'
    },
    children: [{
        type: 'text',
        text: 'Hello World!'
    }]
}]
...
<rich-text nodes="{{nodes1}}" bindtap="tap"></rich-text>
```

nodes1 是一个数组，node 是其元素类型。每个 node 元素都有一个 name、一个 type 属性和一个 attrs 属性。默认 type 等于 node，当 type 等于 node 时，可以有子节点，此时有一个 children 属性。

当 type 等于 node 时，其具有如下属性。

- name：标签名，支持部分受信任的 html 节点。
- attrs：标签的属性，是一个 Object 对象，支持部分受信任的属性，遵循 Pascal 命名法。
- children：子节点数组列表，结构和 nodes 一致。

当 type 等于 text 时，仅有一个 text 属性，此时的节点不包含任何子节点，本身是叶节点。

在上面的代码中，外围标签是 div，内部子标签是 text。这种定义 nodes 的方式是给机器用的，人为手动编写这样的 nodes 是很麻烦的。但如果小程序是一个 html 有限标签编辑器，可动态生成 nodes 标签，那么这样的定义方式就是必需的。从这个意义上来讲，微信小程序成为一个微型浏览器也未可知了。

图 12-3

上述代码的运行效果如图 12-3 所示。

2. 使用 html 字符串定义 rich-text 的 nodes

另一种方式是直接使用 html 字符串，代码如下：

```
nodes2: `<table width="100%" style="text-align:center;box-shadow: 2px 2px 5px #888;">
    <tr>
        <th>Month</th>
        <th>Savings</th>
        <th>third</th>
    </tr>
    <tr>
        <td>January</td>
        <td>$100</td>
        <td>$100</td>
    </tr>
    <tr>
        <td>January1</td>
        <td>$1001</td>
```

```
            <td>$1001</td>
        </tr>
    </table><br>
    <img style="box-shadow: 2px 2px 6px #888;" src="http://www.w3school.com.cn/i/eg_
        tulip.jpg" width="100%" alt="郁金香" />
    <ol>
        <li>Coffee</li>
        <li>Tea</li>
        <li>Milk</li>
    </ol>
    <del>del style</del>,<ins>ins style</ins><br>
    <sub>下标</sub>,<sup>上标</sup>
    <br>
    <em>em标签</em>
    <ul>
        <li>Coffee</li>
        <li>Tea</li>
        <li>Milk</li>
    </ul>
    <h1>这是标题 1</h1>
    <h2>这是标题 2</h2>
    <h3>这是标题 3</h3>
    <h4>这是标题 4</h4>
    <h5>这是标题 5</h5>
    <h6>这是标题 6</h6>
    <hr />
    <fieldset>
        <legend>health information</legend>
        height: <input type="text" />
        weight: <input type="text" />
    </fieldset>
    <p><span>some text.</span>some other text.</p>`
```

这个 html 字符串几乎包括了 rich-text 组件目前可以渲染的所有 html 富文本标签，图 12-4 所示的是其渲染效果。

table、img 标签支持 width、height 属性，所以可以将它们设置为与屏等宽。所有被支持的 html 富文本标签组件，都支持 style 样式和 class 样式。

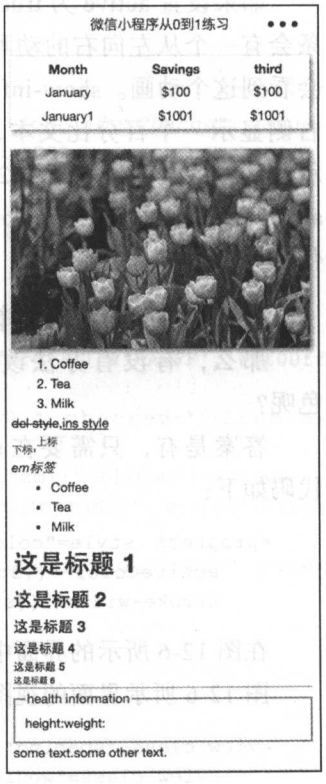


图　12-4

3. 在 wxss 文件中定义 class 以提供给 rich-text 使用

首先，在 rich-text.wxss 文件中定义一个 div_class 样式，代码如下：

```
.div_class{
  font-size: 60px;
}
```

再次测试使用 html 字符串定义 rich-text 的 nodes 的代码

段，运行效果如图 12-5 所示。由此可见，在 wxss 文件中定义的样式可以作用于 rich-text 组件的 node 标签。

图 12-5

12.4 progress

1. progress 组件的属性

微信小程序组件的核心设计准则就是方便开发。基础内容组件 progress 用于显示事务进度，如下所示，官方提供的属性已经可以满足绝大多数的业务逻辑需求。

- percent：百分比 0～100。
- show-info：在进度条右侧显示百分比。
- stroke-width：进度条线的宽度，默认为 6px。
- activeColor：已选择的进度条的颜色。
- backgroundColor：未选择的进度条的颜色。
- active：是否显示进度条从左往右的动画。

图 12-6

如果设置 active 为 true，则在 percent 发生改变时，进度条会有一个从左向右的动画。即使在第一次打开页面时，也会看到这个动画。show-info 属性如果为 true，则会在进度条右侧显示一个百分比文本，如图 12-6 所示。stroke-width 表示进度条的宽度，一般为 2px。color 属性可以忽略，可使用 activeColor 代替。activeColor 代表前景色，backgroundColor 代表背景色。

2. 如何改变百分比文本的颜色

那么，有没有办法改变右边 percent 百分比的数字颜色呢？

答案是有，只需要在 style 属性上添加 color 样式即可，代码如下：

```
<progress style="color:red" percent="30" backgroundColor="{{backgroundColor}}"
    activeColor="{{activeColor}}" show-info="{{showInfo}}" active="{{active}}"
    stroke-width="{{strokeWidth}}" />
```

在图 12-6 所示的界面中，笔者用一个页面测试了几乎所有的属性。

图 12-6 所示界面的视图层源码如下：

```
<view class="spacing">
    <view class="weui-flex">
        <view class="weui-flex__item">
```

```
            <progress style="color:red" percent="30" backgroundColor="{{backgroun
                dColor}}" activeColor="{{activeColor}}" show-info="{{showInfo}}"
                active="{{active}}" stroke-width="{{strokeWidth}}" />
        </view>
    </view>
</view>
<view class="weui-cells">
    <view class="weui-cell weui-cell_switch">
        <view class="weui-cell__bd">显示百分比</view>
        <view class="weui-cell__ft">
            <switch data-name="showInfo" checked="{{showInfo}}" bindchange=
                "onSwitchChange" />
        </view>
    </view>
    <view class="weui-cell weui-cell_switch">
        <view class="weui-cell__bd">进度条动画</view>
        <view class="weui-cell__ft">
            <switch data-name="active" checked="{{active}}" bindchange=
                "onSwitchChange" />
        </view>
    </view>
    <view class="weui-cell weui-cell_input">
        <view class="weui-cell__hd">
            <view class="weui-label">进度条宽度</view>
        </view>
        <view class="weui-cell__bd">
            <slider data-name="strokeWidth" bindchange="onSliderChange" value=
                "{{strokeWidth}}" show-value min="2" max="20" />
        </view>
    </view>
</view>
<view class="weui-cells__title">进度条前景色</view>
<view class="weui-cells weui-cells_after-title">
    <radio-group data-name="activeColor" bindchange="onRadioChange">
        <label class="weui-cell weui-check__label" wx:for="{{['#10AEFF','#ff00ff
            ','#aaAEFF']}}" wx:key="value">
            <radio class="weui-check" value="{{item}}" checked="{{item ==
                activeColor}}"/>
            <view class="weui-cell__bd" style="color:{{item}}">{{item}}</view>
                <view class="weui-cell__ft weui-cell__ft_in-radio" wx:if="{{item
                    == activeColor}}">
                <icon class="weui-icon-radio" type="success_no_circle" size=
                    "16"></icon>
            </view>
        </label>
    </radio-group>
</view>
<view class="weui-cells__title">背景色</view>
<view class="weui-cells weui-cells_after-title">
    <radio-group data-name="backgroundColor" bindchange="onRadioChange">
        <label class="weui-cell weui-check__label" wx:for="{{['#ccc','#eee','#a
```

```
            aa']}}" wx:key="value">
            <radio class="weui-check" value="{{item}}" checked="{{item ==
                backgroundColor}}"/>
            <view class="weui-cell__bd" style="color:{{item}}">{{item}}</view>
                <view class="weui-cell__ft weui-cell__ft_in-radio" wx:if="{{item
                    == backgroundColor}}">
            <icon class="weui-icon-radio" type="success_no_circle" size="16"></
                icon>
            </view>
        </label>
    </radio-group>
</view>
```

图 12-6 所示界面的逻辑层源码如下，在周期函数 onLoad 中调用 setData，设置 strokeWidth、backgroundColor 等页面数据变量的初始值，是为了在默认情况下视图能显示正常的值，而不是零值：

```
let app = getApp()
Page(Object.assign(app.page,{
    data: {},
    onLoad() {
        this.setData({
            strokeWidth: 6,
            backgroundColor:"#ccc",
            activeColor:"#10AEFF",
            strokeWidth:2
        })
    }
}))
```

第 13 章 Chapter 13

表单组件

小程序的表单组件通常是指包括 input、radio、checkbox、label 在内的 11 个表单组件和辅助表单组件，它们足以满足各种常见的基础表单的业务需求。

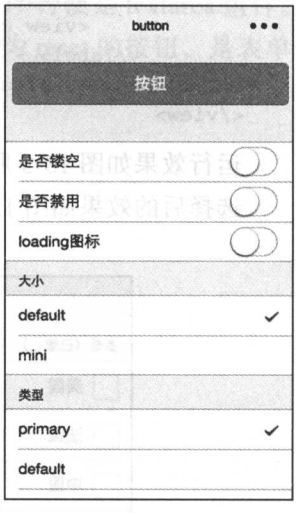


图 13-1

13.1 button

如图 13-1 所示，笔者使用若干表单组件测试了 button 的属性。

1. 当 open-type="getUserInfo" 时

当 open-type="getUserInfo" 时，此时的按钮是一个授权小程序可以拉取用户信息的按钮，示例代码如下。当单击该授权按钮时，如果用户未曾授权过，那么微信会弹窗请求用户授权；如果用户已经授权，则会直接在 onGetUserinfo 方法中返回 userInfo，如图 13-2 所示。

```
<button bindgetuserinfo="onGetUserinfo" open-type=
    "getUserInfo" type="primary">用户授权</button>
```

图 13-2 是用 console.log 打印 event.detail 的截图。

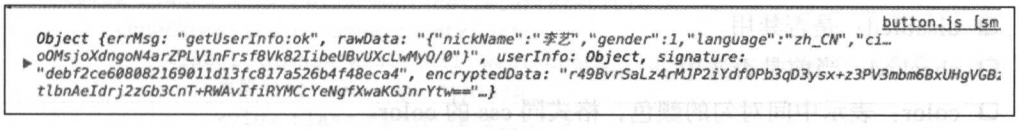


图 13-2

2. 当 open-type="contact" 时

当 open-type="contact" 时，单击后直接进入客服对话窗口。注意，该功能仅能在手机上进行测试，在微信 Web 开发者工具中测试时不能正常工作。

13.2 checkbox

当向用户就某一个问题收集多个答案时，这时可使用多项选择器。

checkbox 是多选项目，单独使用意义不大，一般与多项选择器 checkbox-group 搭配使用。如下代码所示的是一个通用的多项选择器的视图层示例：

```
<view class="weui-cells__title">
    多选（已选：{{countries}}）</view>
<view class="weui-cells weui-cells_after-title">
    <checkbox-group data-name="countries" bindchange="onCheckboxChange">
        <label class="weui-cell weui-check__label" wx:for="{{['美国','法国','中国
            ']}}" wx:key="*this">
            <view class="weui-cell__hd">
                <checkbox color="red" style="color:red" value="{{item}}"/>
            </view>
            <view class="weui-cell__bd">{{item}}</view>
        </label>
    </checkbox-group>
</view>
```

运行效果如图 13-3 所示。

选择后的效果如图 13-4 所示。

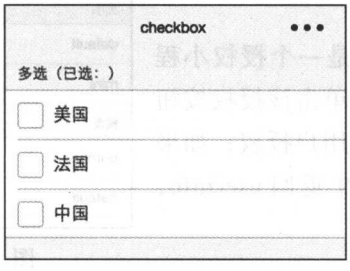


图 13-3

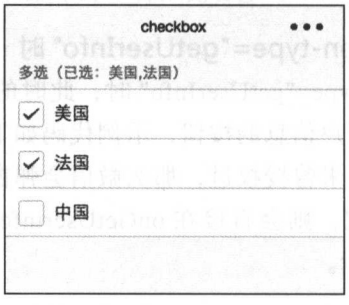


图 13-4

多选项目 checkbox 有如下 4 个属性。

- value：checkbox 的标识，选中时触发 checkbox-group 的 change 事件。
- disabled：是否禁用。
- checked：当前是否选中。
- color：表示中间对勾的颜色，格式同 css 的 color。

属性 value 设置的值体现在 checkbox-group 的选择值中。checkbox-group 的选择值是一

个数组。属性 color 设置的颜色是选择对钩的颜色，如图 13-4 所示。选择框的颜色目前无法定制。

下面的代码是通过 bindchange 绑定的 onCheckboxChange 方法。

```
onCheckboxChange: function (e) {
    var fieldName = e.currentTarget.dataset.name
    var data = {}
    data[fieldName] = e.detail.value
    this.setData(data)
}
```

在 onCheckboxChange 方法中，通过 e.detail.value 获取到的值是一个数组，是所有被选择的选择项目的值。

13.3 form

对于表单，官方文档给出的解释是，将组件内的用户输入的 <switch/> <input/> <checkbox/> <slider/> <radio/> <picker/> 。但事实上，并非只有这 6 个组件可以放在 form 容器内，其他组件（例如 textarea）也可以，代码如下所示，倒数 11 行便是 textarea 组件。formType 属性为 submit 的按钮，是表单提交按钮。formType 属性为 reset 的按钮，是表单重设按钮。

```
<form catchsubmit="formSubmit" catchreset="formReset">
    <view class="weui-cells weui-cells_after-title">
        <view class="weui-cell weui-cell_switch">
            <view class="weui-cell__bd">switch</view>
            <view class="weui-cell__ft">
                <switch name="switch" checked />
            </view>
        </view>
        <view class="weui-cell weui-cell_input">
            <view class="weui-cell__hd">
                <view class="weui-label">input</view>
            </view>
            <view class="weui-cell__bd">
                <input name="input" class="weui-input" placeholder="请输入文本" />
            </view>
        </view>
    </view>
    <view class="weui-cells__title">多项选项器countries</view>
    <view class="weui-cells weui-cells_after-title">
        <checkbox-group name="countries">
            <label class="weui-cell weui-check__label" wx:for="{{['美国','法国','
                中国']}}" wx:key="*this">
                <view class="weui-cell__hd">
                    <checkbox color="red" style="color:red" value="{{item}}"/>
```

```
                </view>
                <view class="weui-cell__bd">{{item}}</view>
            </label>
        </checkbox-group>
    </view>

    <view class="weui-cells__title">滑动选择器slider</view>
    <view class="weui-cells weui-cells_after-title">
        <view class="weui-cell weui-cell_input">
            <view class="weui-cell__hd">
                <view class="weui-label">slider</view>
            </view>
            <view class="weui-cell__bd">
                <slider name="slider" max="10" value="1" show-value/>
            </view>
        </view>
    </view>

    <view class="weui-cells__title">单项选择器color</view>
    <view class="weui-cells weui-cells_after-title">
        <view class="weui-cell__bd">
            <radio-group name="color" data-name="activeColor" bindchange=
                "onRadioChange">
                <label class="weui-cell weui-check__label" wx:for="{{['#10AEFF',
                    '#ff00ff','#aaAEFF']}}" wx:key="value">
                    <radio class="weui-check" value="{{item}}" checked="{{item ==
                        activeColor}}"/>
                    <view class="weui-cell__bd" style="color:{{item}}">{{item}}</
                        view>
                    <view class="weui-cell__ft weui-cell__ft_in-radio" wx:if=
                        "{{item == activeColor}}">
                        <icon class="weui-icon-radio" type="success_no_circle"
                            size="16"></icon>
                    </view>
                </label>
            </radio-group>
        </view>
    </view>

    <view class="weui-cells__title">省市region</view>
    <view class="weui-cells weui-cells_after-title">
        <view class="weui-cell weui-cell_access">
            <view class="weui-cell__bd">
                <picker name="region" mode="region" data-name="region"
                    onRegionPickerChange value="{{['广东省', '广州市', '海珠区']}}">
                    <text wx:if="{{region}}">{{region[0]}},{{region[1]}},{{regi
                        on[2]}}</text>
                <text wx:else>选择</text>
            </picker>
        </view>
```

```
            <view class="weui-cell__ft weui-cell__ft_in-access"></view>
        </view>
        </view>

        <view class="weui-cells__title">文本域textarea</view>
        <view class="weui-cells weui-cells_after-title">
            <view class="weui-cell">
                <view class="weui-cell__bd">
                    <textarea name="textarea" class="weui-textarea" placeholder="请输
                        入文本" style="height: 3.3em" />
                    <view class="weui-textarea-counter">0/200</view>
                </view>
            </view>
        </view>

        <view class="weui-btn-area">
            <button type="primary" formType="submit">
                Submit</button>
            <button formType="reset">Reset</button>
        </view>
    </form>
```

图　13-5

上述代码的运行效果如图 13-5 所示，单击 Submit 按钮，控制台就会打印表单数据；单击 Reset 按钮，所有表单数据就会清零。

1. 一次性获取所有表单数据

单击 Submit 按钮，将会触发 formSubmit 方法，代码如下所示：

```
formSubmit: function (e) {
    console.log('form发生了submit事件，携带数据为：',
        e.detail.value)
}
```

图 13-6 是 formSubmit 函数的输出。

submit 事件携带的 detail.value 是一个字典。键名是表单内输入交互组件的 name。如图 13-7 所示，方框内便是输入交互组件的 name 属性。

```
form发生了submit事件，携带数据为：
▼Object {switch: true, input: "text", countries: Array[2], slider: 1, color: "#ff00ff"…}
    color: "#ff00ff"
  ▶ countries: Array[2]
    input: "text"
  ▶ region: Array[3]
    slider: 1
    switch: true
    textarea: "mutilines"
```

图　13-6

```
<view class="weui-cells__title">省市region</view>
<view class="weui-cells weui-cells_after-title">
  <view class="weui-cell weui-cell_access">
    <view class="weui-cell__bd">
      <picker name="region" mode="region" data-name="region"
      value="{{['广东省', '广州市', '海珠区']}}">
        <text wx:if="{{region}}">{{region[0]}},{{region[1]}},{{region[2]}}</text>
        <text wx:else>选择</text>
      </picker>
    </view>
    <view class="weui-cell__ft weui-cell__ft_in-access"></view>
  </view>
</view>
<view class="weui-cells__title">文本域textarea</view>
<view class="weui-cells weui-cells_after-title">
  <view class="weui-cell">
    <view class="weui-cell__bd">
      <textarea name="textarea" class="weui-textarea" placeholder="请输入文本"
      style="height: 3.3em" />
      <view class="weui-textarea-counter">0/200</view>
    </view>
  </view>
</view>
```

图 13-7

使用表单容器 form，以及相关的 formType 为 submit 的提交按钮和 formType 为 reset 的重设按钮，其意义在于不仅可以一次性获取所有输入数据，还可以一次性清空已输入的所有数据。

2. 在 reset 事件函数中处理意外情况

在准备上述示例的过程中，曾出现了一个意外的情况——单击 Reset 按钮，已选择的 color 并不能清空，如图 13-8 所示。

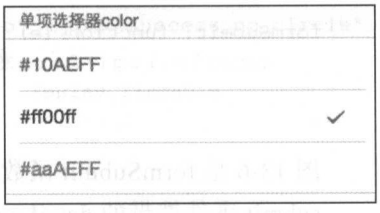


图 13-8

这是因为图 13-8 所示的视觉效果并不是由单项选择器 radio-group 和单项选择组件 radio 决定的，而是使用 view、icon 组件自定义显示的。示例代码如下，粗体部分是自定义选择勾的标签代码。

```
<view class="weui-cells__title">单项选择器color</view>
<view class="weui-cells weui-cells_after-title">
    <view class="weui-cell__bd">
        <radio-group name="color" data-name="activeColor" bindchange=
            "onRadioChange">
            <label class="weui-cell weui-check__label" wx:for="{{['#10AEFF','#ff
                00ff','#aaAEFF']}}"
            wx:key="value">
                <radio class="weui-check" value="{{item}}" checked="{{item ==
                    activeColor}}"/>
```

```
                <view class="weui-cell__bd" style="color:{{item}}">{{item}}</
                    view>
                <view class="weui-cell__ft weui-cell__ft_in-radio" wx:if="{{item
                    == activeColor}}">
                    <icon class="weui-icon-radio" type="success_no_circle"
                        size="16"></icon>
                </view>
            </label>
        </radio-group>
    </view>
</view>
```

要解决这个问题，需要在表单的重设事件函数里处理。笔者通过 catchreset 将重设事件绑定到 formReset 函数。formReset 函数的代码如下：

```
formReset: function (e) {
    console.log('form发生了reset事件')
    this.setData({
        activeColor: ''
    })
}
```

在 formReset 函数中，将页面变量 activeColor 清空，这样就实现了自定义 radio 的选择样式清空。

3. form 的属性

表单容器组件 form 有如下三个属性。其中 bindsubmit 和 bindreset 是事件属性；另一种绑定这两种事件的写法分别是 catchsubmit 和 catchreset。

- report-submit：确定是否返回 formId，可用于发送模板消息 。
- bindsubmit：携带 form 中的数据触发 submit 事件，event.detail = {value : {'name': 'value'} , formId: ''}。
- bindreset：表单重置时会触发 reset 事件。

如下面的代码所示，当 form 为 report-submit 属性时，submit 事件会带一个 detail.formId。formId 将用于模板消息的发送，在这里不作展示描述。

```
formSubmit: function (e) {
    console.log('form发生了submit事件，携带数据为：', e.detail.value)
    console.log('form发生了submit事件，formId：', e.detail.formId)
}
```

13.4 input

每个小程序组件的属性都是针对特定的需求场景而设计的，本节将尝试列举几个 input 组件的场景实例。以下示例会涉及输入法面板，需要在手机上进行测试。

1. 自动聚焦

当用户单击进入新页面时，如何实现将焦点自动定位于某个输入框，并呼出文本输入法面板呢？效果如图 13-9 所示。

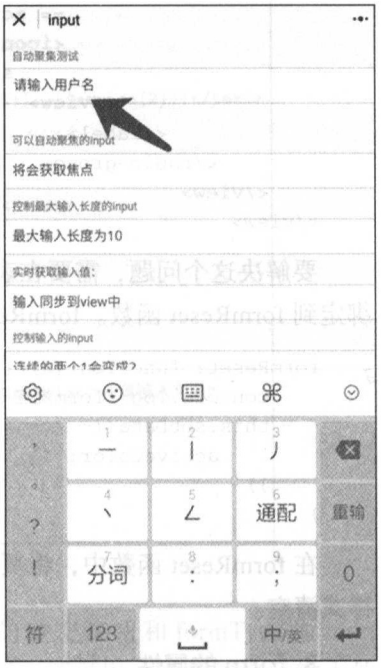


图 13-9

使用 input 组件的 focus 属性可以实现这个功能，代码如下：

```
<input focus type="text" class="weui-input"
    maxlength="140"
placeholder="请输入文本" />
```

在上述代码中，type 属性用于指示呼出的输入法面板类型，共有 4 个有效值，具体如下。

- text：文本输入键盘。
- number：数字输入键盘。
- idcard：身份证输入键盘。
- digit：带小数点的数字键盘。

placeholder 属性是输入内容为空的占位符文本，默认为空。placeholder-style 用于设置占位符的内嵌样式，与一般组件的 style 属性的用法相同。placeholder-class 属性则是以类样式名设置占位符的样式，其用法与一般组件的 class 属性相同。

maxlength 属性用于设置最大输入长度，设置为 -1 的时候不限制最大长度。

在一些旧的小程序教程代码甚至微信官方的示例代码中，还可以看到 auto-focus 属性，它和 focus 属性的作用完全相同，但该属性已经废止，建议用 focus 替代。

每个 input 组件都可以设置 focus 属性，但是每个屏幕只有一个焦点，如果将两个或两个以上的 input 组件设置成了 focus 属性，那么焦点将聚于何处？

答案是聚于最后一个设置 focus 属性的 input 组件之上。最后一个并不一定是 wxml 文件里最下方的一个，如果使用了动态绑定，那么在页面加载之后会动态设置某个或某些 input 组件的 focus 属性，这时就取决于谁是最后设置的了。

2. 密码输入

为了避免旁人看见密码文本，可设置 password 属性使内容以“*”号显示，效果等同于 html 标签的密码输入文本框。代码如下：

```
<input password type="number" class="weui-input" placeholder="请输入密码" />
```

3. 使用单行文本框输入多行文本

使用单行文本框模拟多行文本的输入，效果如图 13-10 所示。

在下面的代码中，上面是视图层标签代码，下面是逻辑层 JS 代码。通过 confirm-hold 属性，可使用户在单击键盘右下角的按钮时保持键盘不收起，confirm-type 属性用于定义右下角按钮的文本（在 Android 上测试无效）。通过 bindconfirm 绑定 onInput1Confirm 方法，即可在每次确定后将新内容追加到页面变量 input1Total 中，并将 input 组件的 value 变量 input1 置空。

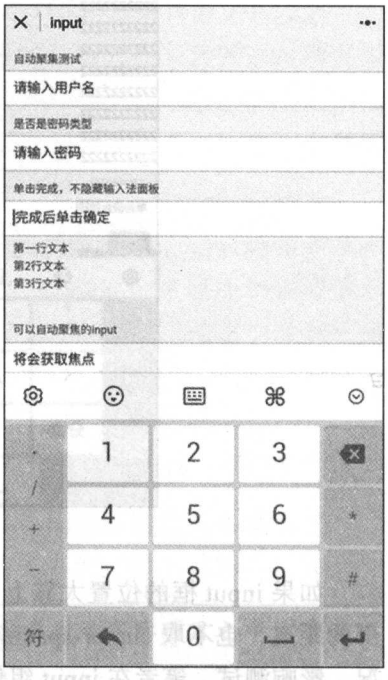


图　13-10

```
<view class="weui-cells__title">单击完成，不隐藏输入法面板</view>
<view class="weui-cells weui-cells_after-title">
    <view class="weui-cell weui-cell_input">
        <view class="weui-cell__bd">
            <input bindconfirm="onInput1Confirm" value="{{input1}}"
confirm-type="确定" confirm-hold class="weui-input" placeholder="完成后单击确定" />
        </view>
    </view>
</view>
<view class="weui-cells__tips"><text> {{input1Total}}</text></view>
...
data: {
    input1Total:'',
    input1:'',
},
onInput1Confirm(e){
    console.log(e.detail.value)
    let newValue = e.detail.value
    if (this.data.input1Total) newValue = this.data.input1Total + "\n" + newValue
    this.setData({
        input1Total: newValue,
        input1:''
    })
}
```

4. 如何扩展输入法面板

当用户单击一个文本框开始输入内容时，能不能在输入法面板上方显示一个颜色拾取器，用于设置文本内容的颜色呢？

答案是可以的，效果如图 13-11 所示。

文本的默认颜色是黑色，单击“颜色”按钮，文本颜色将变为红色。

这个效果是通过 input 组件的 cursor-spacing 属性实现的。

cursor-spacing 属性用于指定光标与键盘的距离，单位为 px 。取 input 距离底部的距离和 cursor-spacing 指定距离的最小值作为光标与键盘的距离，默认值为 0。

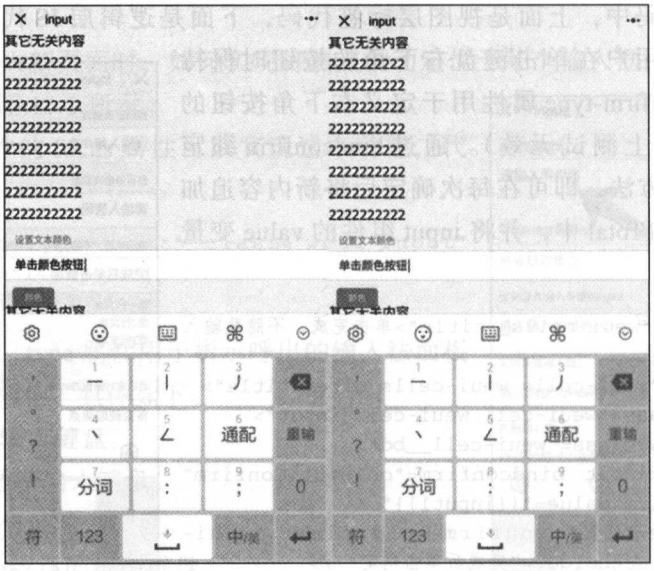


图 13-11

如果 input 框的位置太靠上，与输入法面板相隔很远，则 input 的位置既不取其距离底部的距离，也不取 cursor-spacing 值，而是取决于上边缘距离顶部的距离。为了避免这种情况，影响测试，笔者在 input 组件上方填充了一些“无关内容”。

在下面的代码中，通过绑定 blur 与 focus 事件，分别在 onBlur 和 onFocus 方法中设置了扩展面板的隐藏与显示。当单击“颜色”按钮时，改变页面变量 textColor，即可改变输入框文本的颜色。

```
<view class="weui-cells__title">设置文本颜色</view>
<view class="weui-cells weui-cells_after-title">
    <view class="weui-cell weui-cell_input">
        <view class="weui-cell__bd">
            <input cursor-spacing="50" style="color:{{textColor}}"
bindblur="onBlur" bindfocus="onFocus" class="weui-input" value="单击颜色按钮" />
        </view>
    </view>
    <view class="weui-flex spacing" style="position:absolute;width:100%;display:
        {{inputPanelExtVisible ? 'block' : 'none'}}">
        <view class="weui-flex__item">
            <button bindtap="onTapColor" class="weui-btn mini-btn" type="warn"
                size="mini">颜色</button>
        </view>
    </view>
</view>
<text>
...
data: {
    inputPanelExtVisible:false,
```

```
    textColor:'dark',
},
onTapColor(){
    var color = "red"
    if (this.data.textColor == 'red'){
        color = 'blue'
    }
    this.setData({
        textColor: color
    })
},
onBlur(){
    this.setData({
        inputPanelExtVisible:false
    })
},
onFocus(){
    this.setData({
        inputPanelExtVisible: true
    })
}
```

注意 该效果仅作为场景测试的示例，UI有些粗糙，如果要应用于实际项目之中，读者可以自行尝试优化视觉效果。

13.5 label

在html开发中，有一种功能称为热点功能。在如下代码中，使用area定义了三个热点区域，单击这些区域分别打开不同的页面。area区域是看不见的隐藏区域，所以将它们放在一张图片上面，单击这些区域，实则是单击图片上的特定部位。

```
<img src="img.jpg" border="0" usemap="#map" />
<map name="map">
    <area shape="rect" coords="20,20,40,40" href="about.html" />
    <area shape="circle" coords="80,70,20" href="clearfix.html" />
    <area shape="poly" coords="280,323,323,435,100,300,287,45,34,344" href="position.
        html" />
</map>
```

在小程序中，有没有办法实现类似的热点功能呢？

答案是可以的，使用label组件。

label组件可以扩展目标组件的可单击区域。有两种使用方法：

- 一是使用for属性指定目标组件的id，单击label区域时，即相当于单击目标组件。
- 二是将目标组件作为子标签直接放在label组件内部。

1. 扩展多选项目 checkbox

checkbox 组件本身没有文本，必须借助 label 组件扩展可单击区域。图 13-12 是下面所示代码的运行效果。若不是将 checkbox 组件放在 label 标签的内部，则单击“选项 2”时，第二个多选项目是不会被选中的。

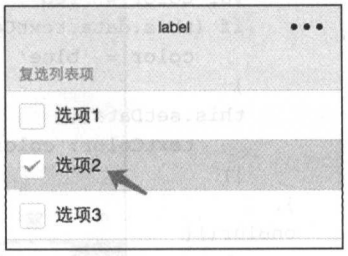


图 13-12

```
<view class="weui-cells weui-cells_after-title">
    <checkbox-group>
        <label class="weui-cell weui-check__label"
            wx:for="{{['选项1','选项2','选项3']}}"
            wx:key="*this">
            <checkbox value="{{item}}"></checkbox>
            <view class="weui-cell__bd">{{item}}</view>
        </label>
    </checkbox-group>
</view>
```

2. 扩展单选项目 radio

同 checkbox 一样，单选项目 radio 同样没有默认标签文本，必须使用 label 组件进行扩展。图 13-13 是下面所示代码的运行效果。扩展项目这两段代码相类似，都是使用 label 组件扩展目标组件的单击区域；不同点在于，扩展单选项目 radio 的代码使用了 label 的 for 属性，一个 label 的 for 属性对应于一个 radio 的 id。

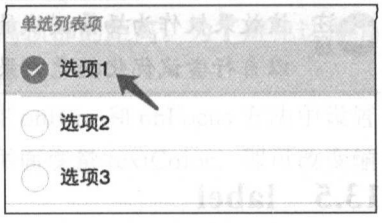


图 13-13

```
<view class="weui-cells__title">单选列表项</view>
<view class="weui-cells weui-cells_after-title">
  <radio-group bindchange="radioChange">
    <view class="weui-cell" wx:for="{{['选项1','选
        项2','选项3']}}" wx:key="*this">
      <radio id="radio{{index}}" value="{{item}}"/>
      <label for="radio{{index}}" class="weui-check__label">
          <view class="weui-cell__bd">{{item}}</view>
      </label>
    </view>
  </radio-group>
</view>
```

3. 扩展 switch 组件

扩展 switch 组件与扩展 checkbox、radio 类似，示例代码如下，其运行效果如图 13-14 所示。

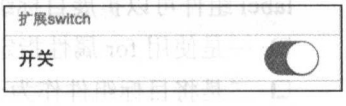


图 13-14

```
<view class="weui-cells__title">扩展switch</view>
<view class="weui-cells weui-cells_after-title">
    <label class="weui-cell weui-cell_switch">
    <view class="weui-cell__bd">开关</view>
        <view class="weui-cell__ft">
```

```
            <switch checked />
        </view>
    </label>
</view>
```

4. 如何实现热点功能

除了扩展 radio、checkbox 和 switch 以外，label 还可以扩展 button 组件，方法与上面的类似。

label 标签内部不仅可以放文本，还可以放画布。在如下代码中，在 label 标签内部声明了一个 id 为“firstCanvas”的 canvas。在页面周期函数 onReady 中，使用小程序接口 wx.createCanvasContext 创建一个绘图上下文对象 ctx，接着在 ctx 对象上绘制一个黑三角形，运行效果如图 13-15 所示。

```
<label for="switch1">
    <canvas style="width: 400px; height: 500px;" canvas-id="firstCanvas"></canvas>
</label>
...
onReady(e) {
    var ctx = wx.createCanvasContext('firstCanvas')
    ctx.moveTo(10, 10)
    ctx.lineTo(100, 10)
    ctx.lineTo(100, 100)
    ctx.fill()
    ctx.draw()
}
```

在上面的代码中，指定了 label 的 for 属性为“switch1”，当单击图 13-15 中的黑三角形时，图示中的开关也随之变化。使用小程序的绘图 API 可以绘制不同的形状，当把形状绘制为透明并覆盖放在一张图片上时，就是热点功能。

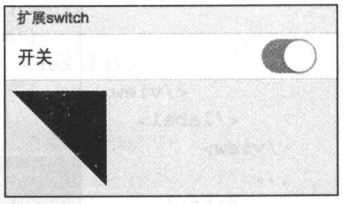


图　13-15

13.6　picker

picker 是从底部弹起的滚动选择器，目前支持 5 种选择器（通过 mode 属性来区分），分别是单列选择器、多列选择器、时间选择器、日期选择器和省市区选择器。

mode 属性的有效值范围如下。

- 单选：selector。
- 多选：multiSelector。
- 时间：time。
- 日期：date。
- 省市：region。

mode 的默认值是 selector。

multiSelector是选择器的底层模式，selector是其一维数组的简化模式，time、date和region是其特定场景下具有特定数据结构的特殊模式。

为什么这样讲？看了多列选择器的使用示例就明白了。

1. 多列选择器

在下面的代码中，展示了一个mode为multiSelector的多列选择器，运行效果如图13-16所示。多列选择器有一个名为columnchange的事件，当多列中的一列选择项发生改变时，该事件将被触发，其event.detail等于{column: column, value: value}。其中，onPickerColumnChange是绑定在columnchange事件上的函数，当column等于0时，即左1列选择变化时，会动态修改左2列绑定的数据multiArray[1]，这时左2列的视图即发生改变。

其他模式的选择器，像时间选择器、日期选择器、省市选择器，都是动态变化数据的多列选择器。微信团队为了简便开发，方便开发者使用，已经将组件场景化封装在了特定场景化的逻辑代码中。

```
<view class="weui-cells__title">多项选择器</view>
<view class="weui-cells weui-cells_after-title">
    <label class="weui-cell">
        <view class="weui-cell__hd">
            <view class="weui-label">当前选择</view>
        </view>
        <view class="weui-cell__bd">
            <picker mode="multiSelector" data-name="multiIndex"
bindcolumnchange="onPickerColumnChange"
bindchange="onPickerChange" range="{{multiArray}}">
                <view>{{multiArray[0][multiIndex[0]]}}, {{multiArray[1][multiIndex[1]]}},
                    {{multiArray[2][multiIndex[2]]}}</view>
            </picker>
        </view>
    </label>
</view>
...
    data:{
        array: ['中国', '美国', '巴西', '日本'],
        objectArray: [
            {
                id: 0,
                name: '美国'
            },
            {
                id: 1,
                name: '中国'
            },
            {
                id: 2,
                name: '巴西'
        },
        {
```

```
            id: 3,
            name: '日本'
            }
        ],
        multiArray: [['无脊柱动物', '脊柱动物'], ['扁性动物', '线形动物', '环节动物', '
            软体动物', '节肢动物'], ['猪肉绦虫', '吸血虫']]
    },
    onPickerColumnChange(e){
        let column = e.detail.column
        let newValue = e.detail.value
        let multiArray = this.data.multiArray

        if (column == 0) {
        if (newValue == 0){
            multiArray[1] = ['扁性动物', '线形动物', '环节动物', '软体动物', '节肢动物']
        }else{
            multiArray[1] = ['有颌鱼', '总鳍鱼', '两栖动物', '爬行动物', '鸟类','哺乳动物']
        }
        this.setData({
            "multiArray": multiArray
        })
    }
},
onPickerChange: function (e) {
    var fieldName = e.currentTarget.dataset.name
    var data = {}
    data[fieldName] = e.detail.value
    this.setData(data)
}
```

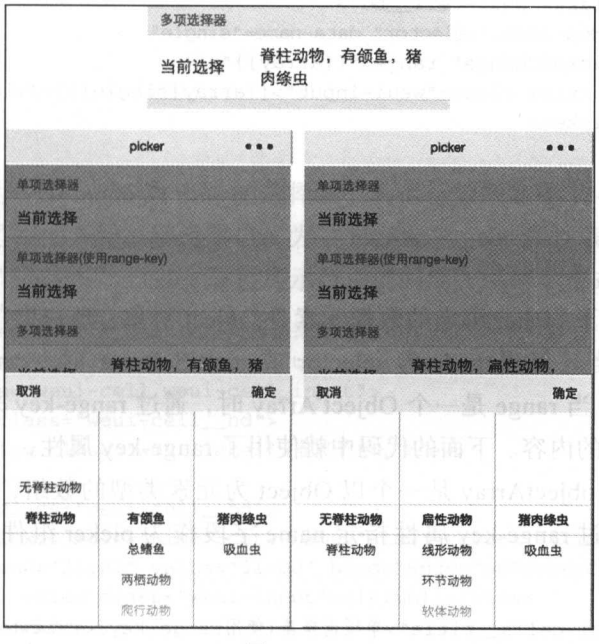


图 13-16

在上面的代码中，picker 组件有一个通用的 change 事件函数 onPickerChange。这个函数的作用就是以 picker 组件上名为 name 的 dataset 值作为变量名，将 picker 的 change 事件的变化值存储到页面数据对象 data 中。下面讲到的选择器，如单列选择器、时间选择器等，都有用到这个函数，使用方法也是一样的。这个函数是通用的编程模式，将它复制一下就可以放在使用 picker 组件的任何项目之中，它已被包含在 sim.js 类库中。关于 sim.js 类库的使用，参见 2.1.4 节的相关内容。

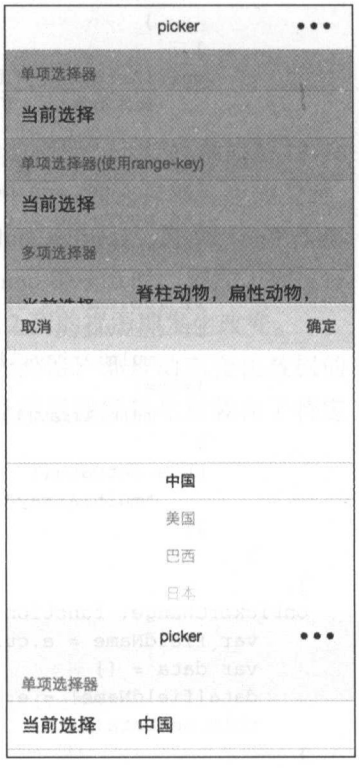


图 13-17

2. 单列选择器

下面代码所示的是 mode 为 selector 的单列普通选择器，array 是绑定在该组件上的页面数据。单列选择器的 change 事件的 event.detail 等于 {value: value}，其中 value 是 array 的索引值，并非元素值，所以在视图上 array[single] 代表已选择的元素值。运行效果如图 13-17 所示。

```
<view class="weui-cells__title">单项选择器</view>
<view class="weui-cells weui-cells_after-title">
    <view class="weui-cell weui-cell_input">
        <view class="weui-cell__hd">
            <view class="weui-label">当前选择</view>
        </view>
        <view class="weui-cell__bd">
            <picker mode="selector" data-name="single"
bindchange="onPickerChange" range="{{array}}">
                <view class="weui-input">{{array[single]}}</view>
            </picker>
        </view>
    </view>
</view>
...
array: ['中国', '美国', '巴西', '日本']
```

考虑到多数情况下 picker 绑定的数据元素是 Object 对象，所以微信团队添加了另外一个属性 range-key。当且仅当 mode 为 selector 或 multiSelector 时，range 属性可用，同时 range-key 属性可用。当 range 是一个 Object Array 时，通过 range-key 来指定 Object 中 key 的值作为选择器显示的内容。下面的代码中就使用了 range-key 属性。

在如下代码中，objectArray 是一个以 Object 为元素类型的数组，每个元素都有 id 和 name 两个字段。通过 range-key 属性指定 name 字段作为 picker 组件在选择器中显示的文本。

```
<view class="weui-cells__title">单项选择器(使用range-key)</view>
<view class="weui-cells weui-cells_after-title">
```

```
  <view class="weui-cell weui-cell_input">
    <view class="weui-cell__hd">
      <view class="weui-label">当前选择</view>
    </view>
    <view class="weui-cell__bd">
      <picker mode="selector" data-name="single2"
bindchange="onPickerChange" range="{{objectArray}}" range-key="name">
        <view class="weui-input">{{objectArray[single2].name}}</view>
      </picker>
    </view>
  </view>
</view>
...
objectArray: [
  {
    id: 0,
    name: '美国'
  },
  {
    id: 1,
    name: '中国'
  },
  {
    id: 2,
    name: '巴西'
  },
  {
    id: 3,
    name: '日本'
  }
]
```

3. 时间选择器

如下代码所示为一个 mode 为 time 的选择器，其运行效果如图 13-18 所示。其中，start 属性表示有效时间范围的开始，字符串格式为“hh:mm”。end 属性表示有效时间范围的结束，字符串格式为“hh:mm”。value 属性表示选中的时间，字符串格式为“hh:mm”。

```
<view class="weui-cells__title">时间选择器</view>
<view class="weui-cells weui-cells_after-title">
    <view class="weui-cell weui-cell_input">
        <view class="weui-cell__hd">
            <view class="weui-label">当前选择</view>
        </view>
        <view class="weui-cell__bd">
            <picker mode="time" data-name="time" value="{{time}}"
start="09:01" end="21:01" value="01:00" bindchange="onPickerChange">
                <view class="weui-input">{{time}}</view>
            </picker>
        </view>
```

```
    </view>
</view>
```

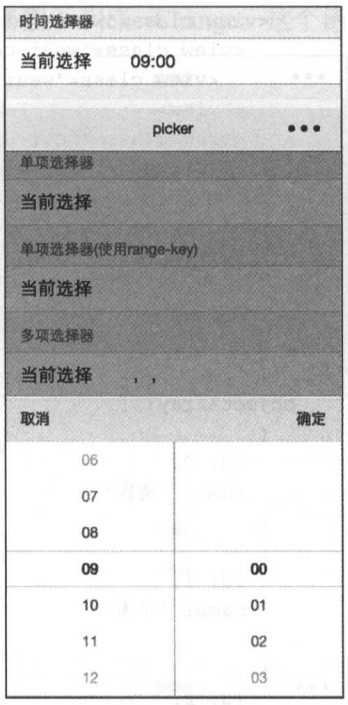


图 13-18

4. 日期选择器

如下代码所示的是一个日期选择器。value 属性表示选中的日期，格式为“YYYY-MM-DD”。start 表示有效日期范围的开始，字符串格式为“YYYY-MM-DD”。end 表示有效日期范围的结束，字符串格式为“YYYY-MM-DD”。其运行效果如图 13-19 所示。

```
<view class="weui-cells__title">日期选择器</view>
<view class="weui-cells weui-cells_after-title">
    <view class="weui-cell weui-cell_input">
        <view class="weui-cell__hd">
            <view class="weui-label">当前选择</view>
        </view>
        <view class="weui-cell__bd">
            <picker mode="date" data-name="date"
                value="{{date}}"
start="2015-09-01" end="2017-09-01" value="2015-09-
02" bindchange="onPickerChange">
                <view class="weui-input">{{date}}</view>
            </picker>
        </view>
    </view>
</view>
```

fields 属性表示选择器的粒度，默认为“year, month, day”。如果改成“year,month”，那么选择器视图将只显示年、月两列。

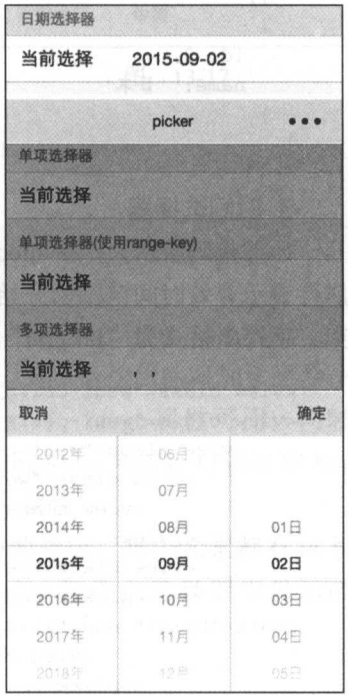


图 13-19

5. 省市选择器

如下代码所示的是一个省市选择器。value 属性表示选中的省、市、区三项，是一个数组。其运行效果如图 13-20 所示。

```
<view class="weui-cells__title">省市选择器</view>
<view class="weui-cells weui-cells_after-title">
    <picker mode="region" data-name="region"
value="{{['广东省', '广州市', '海珠区']}}" bindchange=
"onPickerChange">
        <label class="weui-cell">
        <view class="weui-cell__hd">
            <view class="weui-label">当前选择</view>
        </view>
        <view class="weui-cell__bd">
            <view>{{region}}</view>
```

```
        </view>
    </label>
    </picker>
</view>
```

微信团队在省市选择器内部封装了国内的省市数据，当左侧第 1 列选择的省发生变化时，将自动触发第 2 列市的数据变化，是一个典型的三级多列选择器。

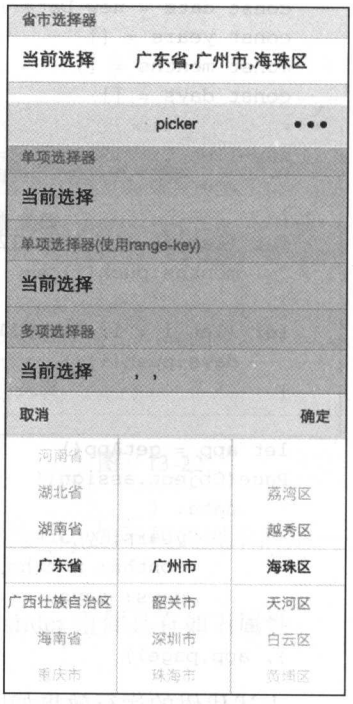


图　13-20

13.7　picker-view

picker-view 是嵌入页面的滚动选择器，picker 组件基于 picker-view 组件实现，当单击 picker 视图时，屏幕下方将弹出一个 picker-view 选择器。picker-view 既可内用于 picker 组件之中，也可以单独使用。本节将展示单独搜索 picker-view 组件的使用场景。

在下面的代码中，上面是视图层代码，下面是逻辑层代码。picker-view 组件的 value 属性是一个数字数组，数组中的数字依次表示 picker-view 内的 picker-view-colume 选择的第几项。indicator-style 属性用于设置选择器中间选中框的样式，indicator-class 用于设置选择器中间选中框的样式类名，这两个属性在开发中很少用到。

需要说明的是，在“let app = getApp()”这一行上面的代码是用于初始化页面数据的。为了简单起见，各月均取了 31 天。

```
<view class="weui-cells__title">当前日期 {{years[value[0]]}}年
{{months[value[1]]}}月{{days[value[2]]}}日</view>
<view class="weui-cells weui-cells_after-title">
    <picker-view style="width: 100%; height: 200px;background-color:#bfc8bc"
value="{{[1,1,1]}}" data-name="value" bindchange="onPickerChange">
        <picker-view-column>
            <view wx:for="{{years}}" wx:key="*this"
style="text-align:center;line-height: 34px">{{item}}年</view>
        </picker-view-column>
        <picker-view-column>
            <view wx:for="{{months}}" wx:key="*this"
style="text-align:center;line-height: 34px">{{item}}月</view>
        </picker-view-column>
        <picker-view-column>
            <view wx:for="{{days}}" wx:key="*this"
style="text-align:center;line-height: 34px">{{item}}日</view>
        </picker-view-column>
    </picker-view>
```

```
</view>
...
const date = new Date()
const years = []
const months = []
const days = []

for (let i = 1990; i <= date.getFullYear(); i++) {
    years.push(i)
}
for (let i = 1; i <= 12; i++) {
    months.push(i)
}
for (let i = 1; i <= 31; i++) {
    days.push(i)
}

let app = getApp()
Page(Object.assign({
    data: {
        years: years,
        months: months,
        days: days,
    }
}, app.page))
```

上述代码的运行效果如图 13-21 所示。选择框之外的渐变效果是通过设置 picker-view 的 background-color 样式形成的。在 picker-view 组件内部，基于选择框的背景色应用了 CSS 滤镜。

在上述代码的视图层代码中，在 style 属性中设置 picker-view 的 width、height 将会非常重要，如果不设置这两个样式，picker-view 视图将会无法正常显示。

图 13-21

13.8 radio

radio-group 是单项选择器，内部由多个单选项目 <radio/> 组成。单项选择器适用于从一个一维数组中选择一个元素的场景。在下面的代码中，视图层代码展示的是一个单项选择器，其运行效果如图 13-22 所示。

```
<view class="weui-cells__title">单项选择（当前选择：{{country}}）</view>
<view class="weui-cells weui-cells_after-title">
    <radio-group data-name="country" bindchange="onRadioChange">
        <label class="weui-cell weui-check__label"
wx:for="{{['美国','中国','日本','英国']}}" wx:key="*this">
```

```
            <view class="weui-cell__hd">
                <radio value="{{item}}" checked="{{item == country}}"/>
            </view>
            <view class="weui-cell__bd">{{item}}</view>
        </label>
    </radio-group>
</view>
...
let app = getApp()
Page(Object.assign({
    data: {}
}, app.page))
```

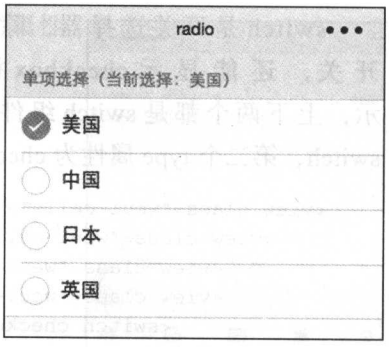


图　13-22

使用类似于图 13-22 这样的 UI，视图简单清晰明了，可将其应用于工具类小程序项目中。

13.9　slider

slider 是滑动选择器，它是小程序组件家族里最简单的组件之一。如下代码展示的是一个 slider 组件，其运行效果如图 13-23 所示。

该段代码的视图层代码中（上部分）就使用了 slider 组件。slider 组件具有如下属性。

- min：表示最小值。
- max：表示最大值。
- step：表示步长，取值必须大于 0，并且可被 (max–min) 整除。
- backgroundColor：表示背景色，即图 13-23 中的灰色。
- activeColor：表示已选择部分的颜色，即图 13-23 中的红色。
- show-value：表示是否显示当前 value。

```
<view class="weui-cells__title">滑动选择器（当前：{{slider}}）</view>
<view class="weui-cells weui-cells_after-title">
    <view class="weui-cell weui-cell_input">
        <view class="weui-cell__hd">
            <view>滑块</view>
        </view>
        <view class="weui-cell__bd">
            <slider activeColor="#eb6587" backgroundColor="#bfc8cc"
value="2" min="0" max="10" step="2" data-name="slider"
bindchange="onSliderChange" show-value/>
        </view>
    </view>
</view>
...
let app = getApp()
Page(Object.assign({
    data: {}
}, app.page))
```

图 13-23 所示的 UI 效果，可直接使用于工具类、效率类小程序项目之中。

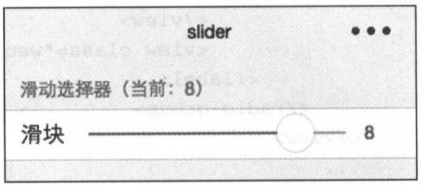


图 13-23

13.10 switch

switch 是开关选择器，除了能显示经典的 iOS 开关，还能显示 checkbox 样式。如图 13-24 所示，上下两个都是 switch 组件，第一个 type 属性为 switch，第二个 type 属性为 checkbox，完整的代码如下所示：

```
<view class="weui-cells">
    <view class="weui-cell weui-cell_switch">
        <view class="weui-cell__bd">switch:{{switch1}}</view>
        <view class="weui-cell__ft">
            <switch checked data-name="switch1" bindchange="onSwitchChange"/>
        </view>
    </view>
    <view class="weui-cell weui-cell_switch">
        <view class="weui-cell__bd">checkbox:{{switch2}}</view>
        <view class="weui-cell__ft">
            <switch color="#eb6587" type="checkbox" data-name="switch2"
bindchange= "onSwitchChange"/>
        </view>
    </view>
</view>
...
let app = getApp()
Page(Object.assign({
    data: {}
}, app.page))
```

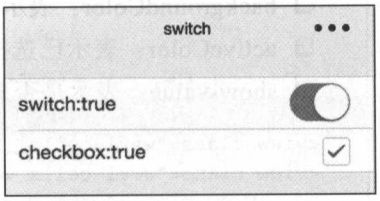


图 13-24

小程序中的 checkbox 组件是作为 checkbox-group 的子组件使用的。对于单个 checkbox 的使用需求，可以考虑使用 switch 组件，设置 type 属性为 checkbox。

13.11 textarea

textarea 是多行输入框，当需要由用户输入多行文本时可使用这个组件。如下代码所示的是 textarea 的用法。其运行效果如图 13-25 所示。

下面的代码中使用了 textarea 组件。textarea 组件主要具有以下属性。

- ❑ auto-height：表示是否自动增高，设置 auto-height 时，style.height 不生效。
- ❑ focus：用于设置是否自动聚集，每个屏幕只能有一个焦点。该属性用于替代旧属性 auto-focus。

- placeholder：表示占位符文本。
- placeholder-style：表示占位符文本的样式属性。
- placeholder-class：用于设置占位符的样式类名。
- max-length：表示最大输入长度，设置为 -1 的时候不限制最大长度。
- cursor-spacing：该属性的作用与 input 组件的同名属性类似，但在实践中，该属性目前还不能正常工作。

```
<view class="weui-cells weui-cells_after-title">
    <view class="weui-cell">
        <view class="weui-cell__bd">
            <textarea class="weui-textarea" style=
            "height:20px" auto-height
            focus placeholder="文本" placeholder-
            style="color:gray"
            maxlength="140" cursor-spacing="0" />
            <view class="weui-textarea-counter">0/200</view>
        </view>
    </view>
</view>
```

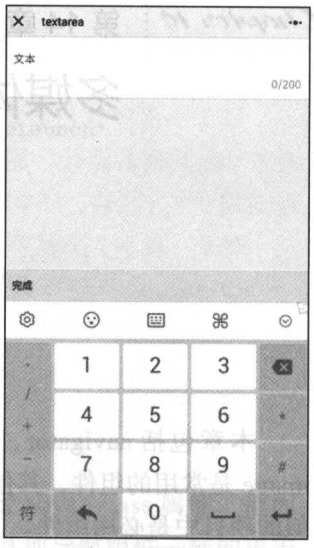
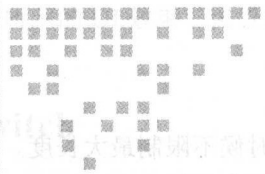


图　13-25

在图 13-25 中，键盘上方的“完成”按钮是小程序默认添加的，单击该按钮将触发 confirm 事件。



Chapter 14 第 14 章

多媒体及其他组件

本章包括 navigator、audio、image、video、map、canvas 6 个组件，其中 navigator 和 image 是常用的组件，其他组件在一般的小程序项目中不会经常用到，但在音视频、LBS 类似的项目中是必用组件。

14.1 navigator

1. 导航组件

navigator 是页面链接组件，相当于 html 标签集中的 a 标签。当需要从一个页面跳转到另一个页面时，使用该组件。navigator 组件根据 open-type 属性的不同，分别对应于 5 个小程序导航 API，具体如下。

- 当 open-type 为 navigate 时，相当于 wx.navigateTo。
- 当 open-type 为 redirect 时，相当于 wx.redirect。
- 当 open-type 为 switchTab 时，相当于 wx.switchTab。
- 当 open-type 为 reLaunch 时，相当于 wx.reLaunch。
- 当 open-type 为 navigateBack 时，相当于 wx.navigateBack。

如下代码所示的是 5 种 open-type 有效值的使用示例。hover-class 属性代表组件被单击时的样式类名，默认值为“navigator-hover”，在 wxss 文件中重写该样式类，可以覆盖默认样式。其运行效果如图 14-1 所示。

```
<view class="weui-btn-area">
    <navigator url="navigator?title=navigate" open-type="navigate"
hover-class="navigator-hover">
```

```
        <button type="default">navigate:新窗口打开</button>
    </navigator>
    <navigator url="navigator?title=redirect" open-type="redirect"
hover-class="navigator-hover">
        <button type="default">redirect:在当前页打开</button>
    </navigator>
    <navigator url="navigator" open-type="switchTab"
hover-class="navigator-hover">
        <button type="default">switchTab:切换tab</button>
    </navigator>
    <navigator url="navigator?title=reLaunch" open-type="reLaunch"
hover-class="navigator-hover">
        <button type="default">reLaunch:重启打开</button>
    </navigator>
    <navigator delta="1" open-type="navigateBack">
        <button type="default">navigateBack:返回页面</button>
    </navigator>
</view>
...
.navigator-hover button{
    background-color: #DEDEDE;
}
```

在上面的代码中，最后一个导航组件 navigator，虽然没有指定 hover-class 属性，但因为在 wxss 中重写了“.navigator-hover”样式类，所以其仍具有和其他导航组件一样的单击状态，如图 14-2 所示。

图　14-1

图　14-2

5 种 open-type 有效值，默认都有 url 属性，仅当 open-type 为 navigateBack 时例外。当 open-type 为 navigateBack 时，url 无效，另有 delta 属性表示反退的级别，默认为 1。

一般来说 url 都可以有页面参数，仅当 open-type 为 switchTab 时，url 属性不带附带页面参数。

如果一个页面被声明为 tab 页，即在 app.json 文件的 tabBar 配置中使用了这个页面的地址，那么这个页面就不能再用 wx.navigateTo 或 wx.redirect 进行跳转，而是只能用

wx.switchTab 进行切换（并且切换时不能附带页面参数），否则会无法实现跳转。

2. 导航接口

小程序有 5 个导航接口，使用导航接口可以在逻辑层实现页面的跳转。

（1）wx.navigateTo(OBJECT)

该接口会在跳转到新页面时保留当前页面。跳转之后，使用 wx.navigateBack 可以返回原页面。

每个小程序接口接收的参数都是 Object 对象。在这个接口参数对象中，url 是要跳转到的非 tab 页面的路径，路径之后可以带参数。参数与路径之间使用“？”分隔，参数键与参数值用“=”相连，不同的参数用“&”分隔，如“path?key=value&key2=value2”。

每个（异步）小程序接口都可以在参数对象中指定三个回调函数，具体如下。

- success：接口调用成功的回调函数。
- fail：接口调用失败的回调函数。
- complete：接口调用结束的回调函数，无论调用成功、失败都会执行。

代码示例如下：

```
wx.navigateTo({
    url: 'newpage?id=1',
    success:res => {},
    fail:err => {},
    complete:_ => {}
})
```

（2）wx.redirectTo(OBJECT)

该接口会关闭当前页面，跳转到新页面。url 等参数与 wx.navigateTo 接口相同。

代码示例如下：

```
wx.redirectTo({
    url: 'otherpage?id=1',
    success:res => {},
    fail:err => {},
    complete:_ => {}
})
```

（3）wx.switchTab(OBJECT)

该接口会跳转到 tabBar 页面，并关闭其他所有非 tabBar 的页面。相当于是 reLaunch 了某个 tab 页面。

url 参数必须是 tab 页面的路径，是在 app.json 的 tabBar 配置字段中定义的页面地址，路径之后不能带参数。

代码示例如下：

```
{
    "tabBar": {
```

```
        "list": [{
            "pagePath": "index",
            "text": "首页"
        }...]
    }
}
...
wx.switchTab({
    url: '/index',
    success:res => {},
    fail:err => {},
    complete:_ => {}
})
```

(4) wx.navigateBack(OBJECT)

该接口会关闭当前页面，返回上一级页面或多级页面。

对于该接口，url 等参数无效。delta 参数指定要返回的页深，默认为 1，即返回上一次打开的页面。

代码示例如下：

```
wx.navigateBack({
    delta: 1
})
```

(5) wx.reLaunch(OBJECT)

该接口会关闭所有页面，打开新页面。

url 参数等同于 wx.navigateTo 接口的 url 参数，但也可以是 tab 页面的路径。

一旦代码调用了这个页面，微信就能确认其他页面不再被依赖，就能将它们销毁、回收，从而提升小程序的使用体验。依此来看，这是一个优化 API。如果小程序项目中有一个“首页”按钮，当要单击“首页”按钮时，不妨使用该接口。

代码示例如下：

```
wx.reLaunch({
    url: 'homepage',
    success:res => {},
    fail:err => {},
    complete:_ => {}
})
```

14.2　audio

1. audio 组件

audio 是音频组件，用于播放一个基于 HTTP 的音频资源。下面的代码展示的是一个基于 audio 组件实现的带有 audio 的默认播放控件，并且可扩展出拖动播放、播放 / 暂停、从

头播放功能的播放器。其运行效果如图 14-3 所示。

在视图层代码中，audio 即为音频标签，poster 属性代表默认控件上的音频封面的图片资源地址，若未启用 controls，则不必设置 poster。name 代表默认控件上的音频名字，author 代表默认控件上的作者名字，controls 代表是否显示默认控件。src 是将要播放音频的资源地址，是必须要有的属性。loop 表示是否循环播放。

id 是 audio 组件的唯一标识符，如果在逻辑层代码中使用接口 wx.createAudioContext 创建音频资源的上下文对象，则必须设置这个属性。

timeupdate 事件，在播放进度发生改变时触发，其 detail = {currentTime, duration}，将该事件绑定到 onTimeUpdate 函数。play 事件在开始 / 继续播放时触发，将该事件绑定到 onPlay 函数。pause 事件，当播放暂停时触发。ended 事件在音频播放到末尾时触发，同时绑定 pause、ended 事件到 onStop 函数。

在逻辑层代码中，使用页面数据变量 playing 标识当前音频是否处于播放状态，并在 onPlay、onStop 函数中维护该变量的状态变化。在视图层代码中，会根据变量 playing 使用 JS 的三目运算符动态渲染播放 / 暂停按钮的 src 属性，代码如下：

```
src="{{!playing ? 'play' : 'pause'}}.png"
```

在页面周期函数 onReady 中，调用 wx.createAudioContext 接口创建了一个音频上下文对象 audioCtx，该对象存储在 this 对象上，即当前的 Page 对象。

```
<view class="spacing">
    <audio style="width:100%" poster="{{audio.poster}}"
    name="{{audio.name}}" author="{{audio.author}}" src="{{audio.src}}"
    id="myAudio" controls loop bindtimeupdate="onTimeUpdate"
    bindplay="onPlay" bindpause="onStop" bindended="onStop"></audio>
</view>
<view class="weui-cells weui-cells_after-title">
    <view class="weui-cell weui-cell_input">
        <view class="weui-cell__hd vcenter">
            <image bindtap="play" src="{{!playing ? 'play' : 'pause'}}.png"
            style="width:28px;height:28px;padding-right:5px;"></image>
        </view>
        <view class="weui-cell__bd">
            <slider min="0" max="{{audio.duration}}" value="{{audio.current}}"
bindchange="seek" />
        </view>
        <view class="weui-cell__ft vcenter">
            <image bindtap="restart" src="replay.png"
style="width:28px;height:28px;"></image>
        </view>
    </view>
</view>
...
Page({
    data: {
```

```
        playing:false,
        audio: {
            current:0,
            duration:0,
            poster:
'http://y.gtimg.cn/music/photo_new/T002R300x300M000003rsKF44GyaSk.jpg?max_
    age=2592000',
            name: '此时此刻',
            author: '许巍',
            src:
'http://ws.stream.qqmusic.qq.com/M500001VfvsJ21xFqb.mp3?guid=ffffffff82def4af4b1
    2b3cd9337d5e7&uin=346897220&vkey=6292F51E1E384E06DCBDC9AB7C49FD713D632D313A
    C4858BACB8DDD29067D3C601481D36E62053BF8DFEAF74C0A5CCFADD6471160CAF3E6A&fromt
    ag=46',
        }
    },
    onReady(e) {
        this.audioCtx = wx.createAudioContext('myAudio')
    },
    onPlay(){
        this.setData({
            "playing": true
        })
    },
    onStop() {
        this.setData({
            "playing": false
        })
    },
    seek(e){
        var cur = Math.round(e.detail.value)
        // console.log(cur)
        this.audioCtx.seek(cur)
    },
    onTimeUpdate(e){
        this.setData({
            "audio.current": Math.round(e.detail.currentTime),
            "audio.duration": Math.round(e.detail.duration)
        })
    },
    play() {
        var playing = this.data.playing
        if (playing){
            this.audioCtx.pause()
        }else{
            this.audioCtx.play()
        }
    },
    restart() {
        this.audioCtx.play()
```

```
            this.audioCtx.seek(0)
        }
    })
```

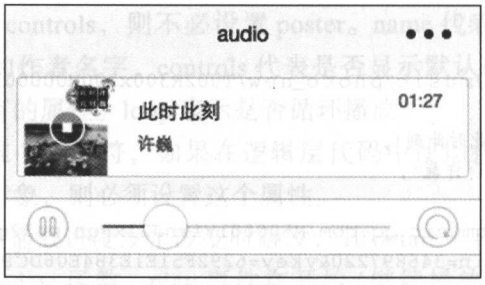


图 14-3

2. AudioContext 对象

AudioContext 对象用于和 audio 组件进行绑定，通过它可以在逻辑层操作视图层的 audio 组件。可以用 wx.createAudioContext（audioId）接口创建，并返回 audio 的上下文对象 AudioContext。

AudioContext 具有以下属性。

- ❑ setSrc：设置音频的新地址，基于该方法可以实现音乐专辑的连续播放。
- ❑ play：播放。
- ❑ pause：暂停。
- ❑ seek：跳转到指定位置，单位为秒。

14.3 image

image 是图像组件，显示本地或网络上的一张图片。src 属性代表图片地址，mode 属性指示图片内容裁剪、缩放的模式，image 相当于是一块画布，当 image 本身的大小与图像内容的大小不一致时，如何将图像内容绘制到画布上？保留哪些舍弃哪些呢？事实上，保留与舍弃的策略就是 mode。mode 共有 13 种，但常用的只有以下 3 种。

- ❑ scaleToFill：不保持纵横比缩放图片，使图片的宽高完全拉伸至填满 image 区域。这种模式适用于以单色小图片作为背景填充图的场景。
- ❑ aspectFit：保持纵横比缩放图片，使图片的长边能够完全显示出来，可以完整地显示图片。这种模式适用于在文章中插图的场景。
- ❑ aspectFill：保持纵横比缩放图片，只保证图片的短边能够完全显示出来。也就是说，图片通常只在水平或垂直方向是完整的，另一个方向将会发生截取。这种模式一般适用于页首的广告图。

另外 10 种不常用的 mode 分别如下。

- widthFix：宽度不变，高度自动变化，保持原图宽高比不变。
- top：不缩放图片，只显示图片的顶部区域。
- bottom：不缩放图片，只显示图片的底部区域。
- center：不缩放图片，只显示图片的中间区域。
- left：不缩放图片，只显示图片的左边区域。
- right：不缩放图片，只显示图片的右边区域。
- top left：不缩放图片，只显示图片的左上边区域（使用时中间有空格）。
- top right：不缩放图片，只显示图片的右上边区域。
- bottom left：不缩放图片，只显示图片的左下边区域。
- bottom right：不缩放图片，只显示图片的右下边区域。

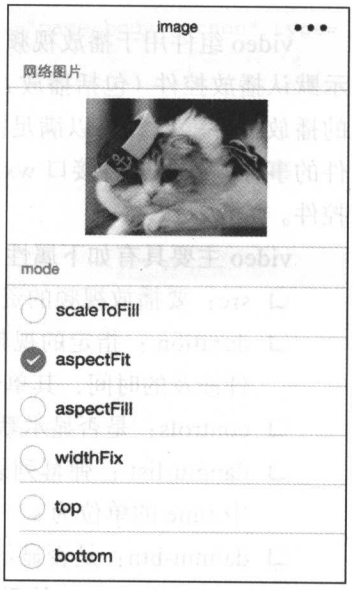


图　14-4

图 14-4 所示的是 13 种模式的测试程序。下面的代码是其逻辑层代码。使用 image 组件需要指定它的大小，宽与高至少指定一个。最常用的情况是这样的，指定 image 的宽度为 100%，设置 mode 为 aspectFit，如图 14-4 中的显示效果所示。

```
<view class="weui-cells__title">网络图片</view>
<view class="weui-cells weui-cells_after-title">
    <view class="weui-cell">
        <view class="weui-cell__bd" style="line-height:0">
            <image style="width:100%;height:300rpx" mode="{{mode}}"
            src="https://mp.weixin.qq.com/debug/wxadoc/dev/image/cat/0.jpg?t=
                2017727" />
        </view>
    </view>
</view>
<view class="weui-cells__title">mode</view>
<view class="weui-cells weui-cells_after-title">
    <radio-group data-name="mode" bindchange="onRadioChange">
        <label class="weui-cell weui-check__label"
wx:for="{{['scaleToFill','aspectFit','aspectFill','widthFix','top','bottom','lef
    t','right','center','top left','top right','bottom left','bottom right']}}"
        wx:key="*this">
            <view class="weui-cell__hd">
                <radio value="{{item}}" checked="{{item == mode}}"/>
            </view>
            <view class="weui-cell__bd">{{item}}</view>
        </label>
    </radio-group>
</view>
```

14.4 video

video 组件用于播放视频。和 audio 一样，video 也有 controls 属性，开启该属性，则显示默认播放控件（包括播放 / 暂停按钮、播放进度、时间等）。在大多数业务场景下，默认的播放控件就已经足以满足需求了。除了使用默认的控件之外，也可以通过监听 video 组件的事件，外加通过接口 wx.createVideoContext 获取的视频上下文对象，实现自定义的 UI 控件。

video 主要具有如下属性。

- src：要播放视频的资源地址，一般为网络地址。
- duration：指定的视频时长，从开关向后计算，相当于视频剪辑，单位是 s（媒体组件涉及的时间，其单位都是 s）。
- controls：是否显示默认播放控件。
- danmu-list：弹幕列表，是一个 Object 数组，Object 的结构为 {text,color,time}，其中 time 的单位为 s。
- danmu-btn：是否显示弹幕按钮。
- enable-danmu：是否展示弹幕，和 danmu-btn 一样，是初始化属性，只在初始化时有效，不能动态变更。
- autoplay：是否自动播放。
- loop：是否循环播放。
- muted：是否静音。

如下代码展示了 video 组件的基础使用方法，其运行效果如图 14-5 所示。

```
<view class="container">
    <view class="page-body">
        <view class="page-section tc">
            <video id="myVideo" duration="120"
src="http://wxsnsdy.tc.qq.com/105/20210/snsdyvideodownload?filekey=3028020101042130
1f02016904025348041 02ca905ce620b1241b726bc41dcff44e00204012882540400&bizid=1023&hy=
SH&fileparam=302c020101042530230204136ffd93020457e3c4ff02024ef202031e8d7f02030f4240
0204045a320a0201000400"
                    binderror="videoErrorCallback" danmu-list="{{danmuList}}"
enable-danmu controls></video>
            <view class="weui-cells">
                <view class="weui-cell weui-cell_input">
                    <view class="weui-cell__hd">
                        <view class="weui-label">弹幕内容</view>
                    </view>
                <view class="weui-cell__bd">
                    <input bindblur="bindInputBlur" class="weui-input"type="text"
placeholder="在此处输入弹幕内容" />
                </view>
            </view>
```

```
        </view>
            <view class="btn-area">
                <button bindtap="bindSendDanmu" class="page-body-button" type=
"primary" formType="submit">发送弹幕</button>
            </view>
        </view>
    </view>
</view>
...
let app = getApp()
Page({
    onReady: function (res) {
        this.videoContext = wx.createVideoContext('myVideo')
    },
    inputValue: '',
        data: {
        src: '',
        danmuList:
            [{
                text: '第 1s 出现的弹幕',
                color: '#ff0000',
                time: 1
            },
            {
                text: '第 3s 出现的弹幕',
                color: '#ff00ff',
                time: 3
            }]
        },
    bindInputBlur: function(e) {
        this.inputValue = e.detail.value
    },
    bindSendDanmu: function () {
        this.videoContext.sendDanmu({
            text: this.inputValue,
            color: app.getRandomColor()
        })
    },
    videoErrorCallback: function(e) {
        console.log('视频错误信息:')
        console.log(e.detail.errMsg)
    }
})
```

图　14-5

14.5　map

map 是地图组件，给出中心点的经纬坐标，就会直接显示一张二维地图。使用 map 组件，可以实现在地图上进行标记、划圈、划线等功能。下面的代码展示了 map 组件的基础

使用方法，其运行效果如图 14-6 所示。

在视图层代码中，会使用到 map 组件的如下属性。

- latitude：代表中心纬度。
- longitude：代表中心经度。
- markers：标记点数组，每个标记点是一个 Object 对象。
- circles：在地图上显示圆圈所使用的数据，是一个数组，数组元素是 circle 数据对象。
- scale：缩放级别，取值范围为 5～18，在图 14-6 中，拖动 slider 组件，将改变 map 的缩放级别。
- polyline：用两个以上的坐标点，依次连线形成线路的数据。
- show-location：显示带有方向的当前定位点。
- include-points：缩放视野需要包含的经纬坐标点数组。

```
<map style="width: 100%; height: 300px;" latitude="{{latitude}}"
    longitude="{{longitude}}" markers="{{markers}}" circles="{{circles}}"
    scale="{{scale}}" polyline="{{polyline}}"
    show-location include-points="{{markers}}" bindtap="onTap">
<cover-view>
    <cover-view class="weui-msg__title hcenter"
        style="background-color:rgba(0,0,0,.3);color:white;padding:10rpx">广州海珠
            万里画行</cover-view>
</cover-view>
<cover-view style="position:absolute;width:100%;bottom:0;display:flex;">
    <cover-image style="width:30px;height:30px" data-location="1"
    bindtap="changeLocation" src="/image/green_tri.png" />
    <cover-image style="width:30px;height:30px" data-location="2"
    bindtap="changeLocation" src="/image/green_tri.png" />
</cover-view>
</map>
<view class="weui-cells weui-cells_after-title">
    <view class="weui-cell weui-cell_input">
        <view class="weui-cell__hd">
            <view>缩放</view>
        </view>
        <view class="weui-cell__bd">
            <slider value="{{scale}}" min="5" max="18" step="1" data-name="scale"
                bindchange="onSliderChange" show-value/>
        </view>
    </view>
</view>
...
let app = getApp()
Page(Object.assign({
    data: {
        latitude: 23.099994,
        longitude: 113.324520,
        markers: [{
```

```
            iconPath: "/image/location.png",
            id: 0,
            latitude: 23.099994,
            longitude: 113.324520,
            width: 50,
            height: 50,
            anchor: { x: .5, y: 0.8 },
            title:'创意图',
            callout: { content:'创意图地标建筑'}
        }, {
            iconPath: "/image/location.png",
            id: 0,
            latitude: 23.112600,
            longitude: 113.324520,
            width: 50,
            height: 50,
            anchor: { x: .5, y: 0.8 }
        }],
        circles: [{
            latitude: 23.099994,
            longitude: 113.324520,
            color: "#ff0000",
            fillColor: "#00000010",
            strokeWidth: 0,
            radius: 120
        }, {
            latitude: 23.112600,
            longitude: 113.324520,
            color: "#ff0000",
            fillColor: "#00000010",
            strokeWidth: 0,
            radius: 100
        }],
        polyline: [{
            points: [{
                latitude: 23.099994,
                longitude: 113.324520
            }, {
                latitude: 23.112600,
                longitude: 113.324520,
            }],
            color: "#ff0000",
            width: 3,
            dottedLine: true,
            arrowLine: true
        }]
    },
    onLoad() {
        this.setData({
            scale: 14
        })
```

```
    },
    onTap(e) {
        console.log(e)
    },
    changeLocation(e) {
        var location = e.currentTarget.dataset.location
        let markers = this.data.markers
        if (location == 1) {
            this.setData({
                latitude: markers[0].latitude,
                longitude: markers[0].longitude
            })
        } else {
            this.setData({
                latitude: markers[1].latitude,
                longitude: markers[1].longitude
            })
        }
    }
}, app.page))
```

图 14-6

在上述代码中，使用 markers 在地图上显示了两个绿色的地图标记，这两个标记是图片，markers 定义了显示它们的坐标。markers 元素主要包含如下属性。

- id：标记点 id，markertap 事件回调会返回此 id，通过监听 markertap 事件可以知道用户单击了哪个标记点。
- latitude：纬度，范围 -90°～90°。
- longitude：经度，范围 -180°～180°。
- title：标注点名。
- iconPath：显示的图标路径。
- rotate：顺时针旋转的角度，范围 0～360°，默认为 0。
- alpha：标注的透明度。
- width：标注的图标宽度，默认为图片的实际宽度。
- height：标注的图标高度，默认为图片的实际高度。
- callout：自定义标记点上方的气泡窗口，对象结构为 {content, color, fontSize, borderRadius, bgColor, padding, boxShadow, display}。
- label：为标记点旁边增加标签，对象结构为 {color, fontSize, content, x, y}。
- anchor：经纬度为标注图标的锚点，默认为底边中点，对象结构为 {x, y}，x 表示横向 (0-1)，y 表示竖向 (0-1)，{x:.5, y:.5} 表示中点。在上面的代码中，使用的 anchor

是 {x: .5, y: 0.8}，是底边中点稍上一点的位置。

在上述代码中，使用 circles、polyline 在地图上绘制了两个圆和一条虚线，如图 14-6 所示。

circles 的元素具有如下属性。

- latitude：纬度，范围 -90°～90°。
- longitude：经度，范围 -180°～180°。
- color：描边的颜色，8 位十六进制表示，后两位表示 alpha 值，如 #000000AA。
- fillColor：填充颜色，8 位十六进制表示，后两位表示 alpha 值，如 #000000AA。
- radius：半径，单位为 px。
- strokeWidth：描边的宽度。

polyline 对象具有如下属性。

- points：经纬度数组，例如 [{latitude: 0, longitude: 0}]。
- color：线的颜色，8 位十六进制表示，后两位表示 alpha 值，如 #000000AA。
- width：线的宽度。
- dottedLine：是否显示为虚线。
- arrowLine：是否带箭头。
- borderColor：线的边框颜色。
- borderWidth：线的厚度。

使用 cover-view 组件（内嵌 cover-image 组件）同样可以在地图上显示图片，也可以实现和 markers 一样的视觉效果。但 cover-view 并不能随地图进行缩放，而 polyline、markers、circles 则是和地图一起缩放的，决定它们的不是 {x,y} 坐标，而是 {longitude, latitude} 经纬坐标。

在图 14-6 中，若分别单击左下角的两个三角形按钮，则会在虚线两端的两个坐标点之间切换地图。

14.6　canvas

canvas 是画布组件，可以监听 touch 事件，可以实现基本的二维几何图形的绘制。如下代码展示了 canvas 组件的使用，其运行效果如图 14-7 所示，可以在画布区域用手指划动涂鸦，单击“清空”按钮清空画布。

```
<canvas style="background-color:#cfcfcf"
    canvas-id="canvas" class="canvas" bindtouchmove="onMove"></canvas>
<view class="weui-btn-area">
    <button bindtap="clear" class="weui-btn" type="primary">清空</button>
</view>
...
```

```
Page({
    data: {},
    track: [],
    dirty: false,
    onReady() {
        this.context = wx.createCanvasContext('canvas')
        this.drawInternal = setInterval(_ => {
            if (!this.dirty) return
            for (var i in this.track) {
                let touch = this.track[i]
                this.drawBall(touch.x, touch.y)
            }
            wx.drawCanvas({
                canvasId: 'canvas',
                actions: this.context.getActions()
            })
            this.dirty = false
        }, 50)
    },
    clear() {
        this.track.length = 0
        this.dirty = true
    },
    onMove(e) {
        var touch = e.touches[0]
        this.track.push(touch)
        this.dirty = true
    },
    drawBall(x, y) {
        let context = this.context
        context.beginPath(0)
        context.arc(x, y, 2, 0, Math.PI * 2)
        context.setFillStyle('#eb6587')
        context.setStrokeStyle('rgba(1,1,1,0)')
        context.fill()
        context.stroke()
    },
    onUnload() {
        clearInterval(this.drawInternal)
    }
})
```

在上述代码的逻辑层代码中，页面周期函数 onReady 中，使用全局 JS 函数 setInterval 注册了一个 50ms 触发一次的定时器。在定时器函数中，首先会循环读取坐标数组 track，并在 CanvasContext 对象上建立绘制路径，然后使用 wx.drawCanvas 接口将路径信息绘制到画布之上。

在上述代码中，track 是画布的数据，把这个数据通过 socket 共享给多个用户，就可以实现在线教学的画板功能。

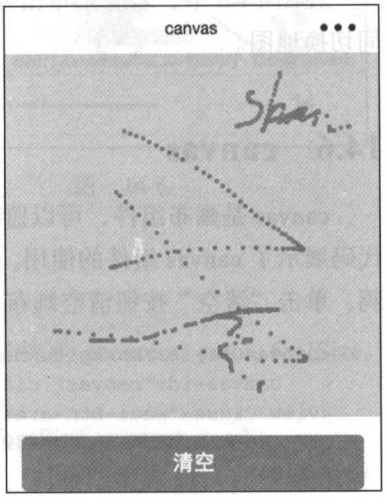
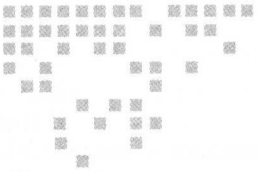


图 14-7

第四篇 Part 4

语言提高篇

- 第 15 章　JavaScript 语言基础
- 第 16 章　WXSS 样式基础
- 第 17 章　Go 语言基础



Chapter 15

第 15 章

JavaScript 语言基础

JavaScript（简称 JS）最早是由 Mozilla 公司的 Brendan Eich 发明的，该语言非常通用，也非常简洁和灵活，已经广泛应用于服务端开发、Web 开发、App 开发等各个领域。JS 也是小程序开发的前端语言，本章将主要介绍该语言的基础语法及一些使用技巧。

15.1 语法基础

针对简短 JS 语法的测试练习，既可以在小程序页面的 onLoad 函数中进行，也可以在微信开发者工具的 Console 面板中直接测试代码，如图 15-1 所示。

```
> [2, 5, 9].forEach((element, index, arr)=>{
      console.log("a[" + index + "] = " + element);
  })
  a[0] = 2
  a[1] = 5
  a[2] = 9
```

图 15-1

15.1.1 变量

变量是存储数据的容器。要声明一个变量，需要使用关键字 var，然后输入你想要的任何名称，比如：

```
var myVariable;
```

行末的分号表示语句结束，不过这个分号只有在单行内需要分割语句时才是必需的，有些人认为在每个语句后面都加上分号是一种好的风格，但笔者认为，如非必要，在小程

序开发中可略去分号。

在小程序里命名变量的名称，第一个字符必须是一个 ASCII 字母（大小写均可），或者一个下划线（_）。注意，第一个字符不能是数字，后续的字符必须是字母、数字或下划线，变量名称一定不能是 JS 语言的关键字、保留字。

一般采用驼峰式命名法为变量命名，变量名或函数名是由一个或多个单词连接在一起组成的，其中第一个单词以小写字母开始，后面所有单词的首字母都采用大写字母，这样的变量名看上去就像驼峰一样此起彼伏，故因此而得名，例如 myVariable。

定义一个变量之后，可以为它赋予一个值：

```
myVariable = 'Bob';
```

可以将这些操作写在一行：

```
var myVariable = 'Bob';
```

可以通过变量名称读取变量：

```
myVariable;
```

在为变量赋值之后，也可以改变变量的值：

```
var myVariable = 'Bob';
myVariable = 'Steve';
```

注意变量包含不同的数据类型，具体如下。

❑ String

字符串，一段文本。若要指示变量是字符串，则应该将它们用引号括起来，例如：

```
var myVariable = 'Bob';
```

❑ Number

数字，一个数字。不用引号括起来，例如：

```
var myVariable = 10;
```

❑ Boolean

布尔型，一个 True/False（真 / 假）值。true/false 是 JS 里的特殊关键字，不需要用引号括起来，例如：

```
var myVariable = true;
```

❑ Array

数组，一种允许在一个引用里存储多个值的结构，例如：

```
var myVariable = [1,'Bob','Steve',10];
```

调用数组的元素只需要这样使用，使用方括号加一个下标数字：

```
myVariable[0], myVariable[1]
```

❑ Object

对象，基本上 JS 里的任何东西都是对象，而且都可以被存储在变量里，例如：

```
var myVariable = {loading:true};
```

在编程时要做任何有趣的事都必须用到变量。如果变量的值没有发生改变，那么将无法动态地做任何事情。

15.1.2 注释

在 JS 代码中添加注释，有两种形式，第一种形式如下所示，其适用于多行注释，一般放在函数上面或源码文件的首部：

```
/*
Everything in between is a comment.
*/
```

第二种是单行注释。如果注释只有一行，那么可将它们更简单地放在两个斜杠之后，就像这样：

```
// This is a comment
```

单行注释既可以放在 JS 语句的上面，又可以放在 JS 语句的后面。

15.1.3 运算符

运算符是一个根据两个值（或变量）做出结果的代数符号。下面将列举一些最简单的运算符，对于后面的示例，可以在浏览器控制台里尝试操作一下。

❑ 加 / 连接：+

用于两个数字的相加，或者连接两个字符串，例如：

```
6 + 9;
"Hello " + "world!";
```

❑ 减、乘、除：-、*、/

这些运算符操作将与它们在基础数学中所做的操作一样，例如：

```
9 - 3;
8 * 2; // JS中的乘是一个"*"号;
9 / 3;
```

❑ 赋值运算符：=

读者之前已经见过这个符号了，它会将一个值赋给一个变量。

```
var myVariable = 'Bob';
```

❑ 相等，全等：==、===

== 将会测试两个值是否相等，而且会返回一个 true/false（布尔型）值。=== 不仅要求

值相等，还要求变量的类型相同。例如：

```
var myVariable = 3;
myVariable === 3;// true
myVariable == '3';// true
```

❑ 非，不等、不全等：!、!=、!==

非运算符“!”经常与相等运算符一起使用。非运算符在 JS 中表示逻辑非，它也返回一个布尔值。在下面的代码中，变量 myVariable 的值是 true，变量 myVariable2 的值是 false：

```
var myVariable = 3;
var myVariable2 = !myVariable === 3;// false
```

不全等运算符“!==”与全等运算符“===”类似，在比较变量值的基础上添加了对变量类型的限制。如下代码所示，myVariable2 的值为 true：

```
var myVariable = 3;
var myVariable2 = myVariable !== '3';// true
```

注意　如果混合几种数据类型可能会导致奇怪的结果，比如输入 "35" + 25，第二个数字会转换成字符串，最终会连接两个字符串而不是数字相加。但如果输入 35 + 25，则会得到正确的结果。

15.1.4　语句

语句是用于测试一个表达式是否返回 true，然后根据结果运行不同代码的结构。最常用的语句形式是 if ... else，下面是一个例子：

```
var iceCream = 'chocolate';
if (iceCream === 'chocolate') {
    this.alert('Yay, I love chocolate ice cream!');
} else {
    this.alert('Awwww, but chocolate is my favorite...');
}
```

if(...) 里面的表达式就是测试——这里使用了运算符（15.1.3 节所提到的）来比较变量 iceCream 与字符串 chocolate 是否相等。如果返回 true，则运行前面一块的代码；如果返回 false，则跳过第一段，直接运行 else 之后的代码。

小程序中是没有 alert 方法的。alert 方法定义在 window 对象上，小程序没有 window 对象，在 Web 中定义在 window 对象上的方法都不能使用。

如果使用了 sim.js 类库的 app.page，则可以使用 this.alert 代替 alert，代码如下所示：

```
let app = getApp()
```

```
Page(Object.assign(app.page, {
    data: {
        outOfBounds:true,
        inertia:true,
        friction:2,
        damping:20,
        direction: "all",
        target:{x:30,y:30}
    },
    onLoad(){
        this.alert(123)
    }
}))
```

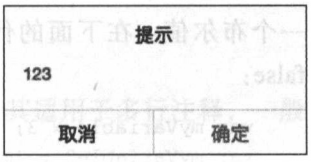


图 15-2

运行效果如图 15-2 所示。

针对 JS 代码的练习，可以在小程序页面的 onLoad 函数内进行。

另一个测试代码的方法是 console.log，它会将信息打印到控制台窗口，因此更加灵活。

15.1.5 函数

函数是一种封装你想要重复使用的功能的方法，在任何时候，若想使用其中的功能都可以通过函数名称来调用，这样就不用重复编写整段代码了。上面已经用过一些这方面的函数了，比如：

```
this.alert('hello!');
```

如果看到一些字符串长得像变量名，但是有括号 () 在后面，那么其可能就是一个函数。函数通常包括一些必要的参数，它们在括号内部，如果有一个以上的参数，则需要使用逗号分开。

比如，this.alert 函数会在页面窗口内弹出一个警告框，但是需要为参数传入一个字符串，以告诉函数应该在警告框里写什么。

用户可以定义自己的函数，下一个例子，会编写一个简单的需要两个参数来做乘法运算的函数：

```
multiply(num1,num2) {
    var result = num1 * num2;
    return result;
}
```

尝试在 onLoad 函数中使用 this.multiply 运行这个函数，然后尝试几次不同的参数，比如：

```
this.multiply(4,7);
this.multiply(20,20);
this.multiply(0.5,3);
```


注意 return 语句用于告诉小程序，返回 result 变量以便后续使用。这是很有必要的，因为函数内定义的变量只能在函数内使用，这称为作用域，简单的理解就是其是变量和函数发挥作用的代码范围。

this 在此处指代当前的页面对象，this.multiply 是调用当前页面作用域内定义的 multiply 方法。在小程序开发中，在 Page 内声明的对象一般只有两种：页面初始数据对象 data 和函数。

15.1.6 事件

在小程序开发中，理解事件非常重要。如同 Web 开发一样，小程序中的事件同样具有冒泡机制。关于冒泡事件，在 3.3.1 节中已讲过。

1. 什么是事件

事件是视图层到逻辑层的通信方式，可以将用户的行为反馈到逻辑层进行处理，将事件绑定在组件上，当事件触发时，就会执行逻辑层中对应的事件处理函数。可以携带额外信息，比如 id、dataset、touches 等。

在之前章节的开发示例中，以 bind 开头的组件属性绑定的均是事件函数，由参数 event 传递过去。

事件又分为冒泡事件和非冒泡事件，具体如下。

- 冒泡事件：当一个组件上的事件被触发后，该事件会向父节点传递。
- 非冒泡事件：当一个组件上的事件被触发后，该事件不会向父节点传递。

前面 3.3.1 节介绍的绑定计算按钮所利用的事件便是冒泡事件。

2. 如何使用事件

首先，在组件中绑定一个事件处理函数。如 bindtap，当用户点击该组件的时候，会在该页面对应的 Page 中找到相应的事件处理函数：

```
<view id="tapTest" data-hi="WeChat" bindtap="tapName"> Click me! </view>
```

其次，在相应的 Page 定义中写上相应的事件处理函数，参数是 event，代码如下：

```
Page({
    tapName: function(event) {
        console.log(event)
    }
})
```

console.log 打印出来的信息大致如下：

```
{
"type":"tap",
"timeStamp":895,
"target": {
```

```
        "id": "tapTest",
        "dataset":  {
            "hi":"WeChat"
        }
    },
    "currentTarget":  {
        "id": "tapTest",
        "dataset": {
            "hi":"WeChat"
        }
    },
    "detail": {
        "x":53,
        "y":14
    },
    "touches":[{
        "identifier":0,
        "pageX":53,
        "pageY":14,
        "clientX":53,
        "clientY":14
    }],
    "changedTouches":[{
        "identifier":0,
        "pageX":53,
        "pageY":14,
        "clientX":53,
        "clientY":14
    }]
    }
```

event 事件的 detail 与 currentTarget.dataset 信息在开发中会经常用到。target 是触发事件的源组件。currentTarget 是事件绑定的当前组件。

3. 如何进行事件绑定

事件绑定的写法与组件的属性相同，以 key=value 的形式书写，具体说明如下。

- key 以 bind 或 catch 开头，后接事件的类型，如 bindtap、catchtouchstart。
- value 是一个字符串，需要在对应的 Page 中定义同名的函数，不然当事件触发的时候会报错。

bind 事件绑定不会阻止冒泡事件向上冒泡，catch 事件绑定可以阻止冒泡事件向上冒泡。

在下面的示例中，点击 inner view 会先后触发 handleTap3 和 handleTap2（因为 tap 事件会冒泡到 middle view，而 middle view 阻止了 tap 事件冒泡，使其不再向父节点传递），点击 middle view 会触发 handleTap2，点击 outter view 会触发 handleTap1。

```
<view id="outter" bindtap="handleTap1">
    outer view
    <view id="middle" catchtap="handleTap2">
```

```
        middle view
        <view id="inner" bindtap="handleTap3">
            inner view
        </view>
    </view>
</view>
```

当组件触发事件时，逻辑层绑定该事件的处理函数会收到一个事件对象。基础事件（BaseEvent）对象包含如下属性。

- type：事件类型。
- timeStamp：事件生成时的时间戳。
- target：触发事件的组件的一些属性值集合。
- currentTarget：当前组件的一些属性值集合。

对于 change 类的 CustomEvent 事件对象，又多了一个 detail 属性，它十分有用，是逻辑层获取视图层信息的主要渠道之一。

TouchEvent 触摸事件对象具有如下属性。

- touches：当前停留在屏幕中的触摸点信息的数组。
- changedTouches：当前变化的触摸点信息的数组。

很少会用到触摸事件的额外属性。

4. 如何自定义及使用数据属性 dataset

在组件中可以自定义数据，这些数据将会通过事件传递给逻辑层。书写方式是以 data- 开头，多个单词之间由连字符“-”进行连接，不能有大写（大写会自动转换成小写），如 data-element-type，最终在 event.currentTarget.dataset 中会将连字符转成驼峰式 elementType。例如，视图层的代码为：

```
<view data-alpha-beta="1" data-alphaBeta="2" bindtap="bindViewTap"></view>
```

逻辑层的代码如下所示，先以 e.currentTarget.dataset 获取到自定义数据的集合对象，再以点访问符获取自定义数据的值：

```
Page({
    bindViewTap:function(e){
        e.currentTarget.dataset.alphaBeta === 1 // - 会转换为驼峰写法
        e.currentTarget.dataset.alphabeta === 2 // 大写会转换为小写
    }
})
```

5. 使用事件的 detail 信息

自定义事件所携带的数据（如表单组件的提交事件）会携带用户的输入，媒体的错误事件会携带错误的信息。点击事件的 detail 对象带有的 x 和 y 属性，代表单击点距离文档左上角的 x 距离和 y 距离。change 事件会带有变化值 value。

15.2 实用的简写技巧

本节所谈及的技巧，在前面的章节中大多数已有使用，掌握这些技巧可以让你看起来更像一个编程高手。但事实上，真正的高手在于思想，并不在于技巧，在使用这些技巧时，须以易写、易读、易维护为前提。

15.2.1 三元操作符

“?:”是三元操作符，因参与运算的对象是三个，所以被称为三元操作符。当使用 if…else 语句时，可以使用更简短的三元操作符来代替，如下代码所示：

```
let x = 20;
let answer;
if (x > 10) {
    answer = 'is greater';
} else {
    answer = 'is lesser';
}
```

可将代码简写为：

```
let answer = x > 10 ? 'is greater' : 'is lesser'
```

15.2.2 逻辑并操作符

连续的两个竖杠是逻辑并操作符。当对一个变量赋值时，若想确定原始值是不是 null、undefined 或空值，可采用如下代码：

```
if (var1 !== null || var1 !== undefined || var1 !== '') {
    var2 = var1;
}else{
    var2 = 'new'
}
```

使用逻辑并操作符的简写方法为：

```
var2 = var1 || 'new'
```

15.2.3 单行声明变量

对于多行声明变量的代码，例如：

```
let x;
let y;
let z = 3;
```

可以简写为：

```
let x, y, z=3;
```

如果声明语句过长，那么此法须慎用。

15.2.4 在 if 语句中使用布尔值

下面的代码：

```
if (var1=== true)
```

可简写为：

```
if (var1)
```

但只有 var1 是真值时，二者语句才相等。

如果判断值不是真值，则可以采用如下代码：

```
let a;
if ( a !== true ) {
    //...
}
```

简写后则为：

```
let a;
if ( !a ) {
    //...
}
```

15.2.5 for 循环

下面的代码：

```
for (let i = 0; i < allImgs.length; i++)
```

可简写为：

```
for (let index in allImgs)
```

也可以使用 Array.forEach：

```
[2, 5, 9].forEach((element, index, arr)=>{
    console.log("a[" + index + "] = " + element);
})
// 输出:
// a[0] = 2
// a[1] = 5
// a[2] = 9
```

15.2.6 短路评价

为变量 dbHost 赋值先使用 if 语句检查 process、env、DB_HOST 其值是否为空值，如

下代码所示：

```
let dbHost;
if (process.env.DB_HOST) {
    dbHost = process.env.DB_HOST;
} else {
    dbHost = 'localhost';
}
```

这种情况可以简写为：

```
const dbHost = process.env.DB_HOST || 'localhost';
```

15.2.7 十进制指数

当需要输入的数字带有很多零时（如 10000000），可以采用指数（1e7）来代替这个数字，比如：

```
for (let i = 0; i < 10000000; i++) {}
```

可简写为：

```
for (let i = 0; i < 1e7; i++) {}
```

下面的语句都是返回 true：

```
1e0 === 1;
1e1 === 10;
1e2 === 100;
1e3 === 1000;
```

15.2.8 对象属性

如果属性名与 key 名相同，则可以采用 ES6 的方法将属性名略去，例如：

```
const obj = { x:x, y:y };
```

可将代码简写为：

```
const obj = { x, y };
```

15.2.9 箭头函数

虽然传统函数的编写方法更易于理解和编写，但其不利于用在嵌套函数之中，如下代码所示：

```
function sayHello(name) {
    console.log('Hello', name);
}
setTimeout(function() {
```

```
    console.log('Loaded')
}, 2000);
list.forEach(function(item) {
    console.log(item);
});
```

使用箭头函数，可简写为：

```
sayHello = name => console.log('Hello', name);
setTimeout(_ => console.log('Loaded'), 2000);
list.forEach(item => console.log(item));
```

15.2.10　隐式返回值

通常使用 return 语句来返回函数的最终结果，一个单独语句的箭头函数能隐式返回其值，此时 return 关键字可以省略。省略 return 关键字的同时，函数必须省略 {}。例如以下代码：

```
function calcCircumference(diameter) {
    return Math.PI * diameter
}
var func = function func() {
    return { foo: 1 };
};
```

可简写为：

```
calcCircumference = diameter => (
    Math.PI * diameter;
)
var func = () => ({ foo: 1 });
```

15.2.11　参数的默认值

为了向函数中的参数传递默认值，通常使用 if 语句来编写代码，但是使用 ES6 定义默认值会更简洁一些，比如以下代码：

```
function volume(l, w, h) {
    if (w === undefined)
        w = 3;
    if (h === undefined)
        h = 4;
    return l * w * h;
}
```

可简写为：

```
volume = (l, w = 3, h = 4 ) => (l * w * h);
volume(2) //output: 24
```

15.2.12 模板字符串

传统的 JS 语言，其输出模板通常是这样写的：

```
const welcome = 'You have logged in as ' + first + ' ' + last + '.'
const db = 'http://' + host + ':' + port + '/' + database;
```

ES6 可以使用反引号和 ${} 对其进行简写，代码如下：

```
const welcome = `You have logged in as ${first} ${last}`;
const db = `http://${host}:${port}/${database}`;
```

15.2.13 解构赋值

下面的代码：

```
const store = this.props.store;
const entity = this.props.entity;
```

可简写为：

```
const { store, entity } = this.props;
```

在解构赋值时，可以修改默认的变量名，如下代码所示，第二个变量名为 contact，代替了默认的 entity：

```
const { store, entity:contact } = this.props;
```

15.2.14 多行字符串

若要输出多行字符串，可使用 (+) 来拼接，如下代码所示：

```
const lorem = 'Lorem ipsum dolor sit amet, consectetur\n\t'
    + 'adipisicing elit, sed do eiusmod tempor incididunt\n\t'
    + 'irure dolor in reprehenderit in voluptate velit esse.\n\t'
```

但若使用反引号，则可以达到简写的作用，如下代码所示：

```
const lorem = `Lorem ipsum dolor sit amet, consectetur
    adipisicing elit, sed do eiusmod tempor incididunt
    irure dolor in reprehenderit in voluptate velit esse.`
```

15.2.15 扩展运算符

连续三个点表示扩展运算符，如下代码所示，前两行代码是复制数组，后两行代码是复制数组，这两个场景都可以使用扩展运算符进行简写：

```
// 连接数组
const odd = [1, 3, 5];
const nums = [2 ,4 , 6].concat(odd);
```

```
// 复制数组
const arr = [1, 2, 3, 4];
const arr2 = arr.slice()
```

可简写为：

```
// 连接数组
const odd = [1, 3, 5 ];
const nums = [2 ,4 , 6, ...odd];
console.log(nums);          // 输出[ 2, 4, 6, 1, 3, 5 ]
// 复制数组
const arr = [1, 2, 3, 4];
const arr2 = […arr];
console.log(arr2);          // 输出[ 1, 2, 3, 4 ]
```

若要在一个数组的任意处插入另一个数组，也可以使用扩展运算符，如下代码所示：

```
let arr = [1, 3, 5 ];
let nums = [2, ...arr, 4 , 6];
console.log(nums)           // 输出[2, 1, 3, 5, 4, 6]
```

比上面的方法稍微麻烦一点的方法是这样的：

```
let arr = [1, 3, 5 ];
let nums = [2, 4 , 6];
nums.splice(1,0,…arr);// splice函数用于在数组的指定位置删除N个元素，并补入新元素
console.log(nums) // 输出[2, 1, 3, 5, 4, 6]
```

也可以使用扩展运算符进行解构，如下代码所示：

```
const { a, b, ...z } = { a: 1, b: 2, c: 3, d: 4 };
console.log(a) // 输出1
console.log(b) // 输出2
console.log(z) // 输出{ c: 3, d: 4 }
```

15.2.16　强制参数

在 JS 中，如果没有向函数参数传递值，则参数为 undefined。为了增强参数赋值，可以使用 if 语句来抛出异常，或者使用强制参数简写方法。例如：

```
function foo(bar) {
    if(bar === undefined) {
        throw new Error('参数缺失异常');
    }
    …
    return bar;
}
```

上面的代码是通过 if 判断抛出异常，下面的代码则是使用强制参数的简写示例，其中使用箭头函数略写的函数变量 UndefinedException 是通用的，可以用于任何一个需要检测参数必要性的函数形参列表中。

```
UndefinedException = () => {
    throw new Error('参数缺失异常');
}
foo = (bar = UndefinedException()) => {
    ...
    return bar;
}
```

JS 是弱安全类型语言，通过这个技巧，可以提高代码的安全性。

15.2.17 新数组函数 find

如果想从数组中查找某个值，通常需要用到循环，如下代码所示，自定义实现的 findDog 函数用于从数组 pets 中查找 type 为 Dog、name 为 Tommy 的元素：

```
const pets = [
    { type: 'Dog', name: 'Max'},
    { type: 'Cat', name: 'Karl'},
    { type: 'Dog', name: 'Tommy'},
]
function findDog() {
    for(let i = 0; i<pets.length; ++i) {
        if(pets[i].type === 'Dog' && pets[i].name === 'Tommy') {
            return pets[i];
        }
    }
}
```

但在 ES6 中，find() 函数能实现同样功能。上面代码中的 findDog 可简写为：

```
pet = pets.find(pet => pet.type ==='Dog' && pet.name === 'Tommy');
console.log(pet); // 输出为{ type: 'Dog', name: 'Tommy' }
```

简写后的代码更加易读。find 是 JS ES6 语法内在支持的数组函数，在上面的代码中，它接收了一个使用箭头函数略写、同时又因为是单行代码而略去了 return 关键字的匿名函数。

15.2.18 双重非位运算操作符

如下代码所示，Math.floor 函数用于向下取整，运行结果为 4：

```
if  (Math.floor(4.9) == 4){
    ...
}
```

使用双重非（~~）位运算操作符，可以代替 Math.floor()，其优势在于运算更快、代码更简洁，代码示例如下所示：

```
if  (~~4.9 == 4){
    ...
}
```

第 16 章 Chapter 16

WXSS 样式基础

CSS 是一种向用户指定文档应如何呈现的语言——它会指定文档的样式、布局等。微信小程序的 WXSS 与 Web 开发中的 CSS 具有相同的语法。

WXSS（WeiXin Style Sheets）是一套样式语言，用于描述 WXML 的组件样式。WXSS 可决定 WXML 的组件应该如何显示。WXSS 具有 CSS 的大部分特性，微信团队主要对 CSS 进行了两方面的扩展：

- 尺寸单位。
- 样式导入。

掌握好基础的 CSS 语法之后，再对微信团队扩展的部分加以了解，就能应对所有小程序开发中的样式问题了。

16.1 语法基础

在微信小程序中，定义在 app.wxss 中的样式为全局样式，其作用于每一个页面。在 page 的 wxss 文件中定义的样式为局部样式，其只作用在对应的页面，并且会覆盖 app.wxss 中相同的选择器。

16.1.1 尺寸单位 rpx

rpx（responsive pixel）可以根据屏幕宽度进行自适应调整。通常，规定的屏幕宽度为 750rpx，比如，在 iPhone6 上，屏幕宽度为 375px，共有 750 个物理像素，则 750rpx=375px=750 物理像素，1rpx=0.5px=1 物理像素。微信建议开发小程序时设计师可以用 iPhone6 作为视觉稿的标准。

16.1.2 样式导入

使用 @import 语句可以导入外联样式表，@import 后跟需要导入的外联样式表的相对路径，用分号（;）表示语句结束。示例代码如下：

```
/** common.wxss **/
.small-p {
    padding:5px;
}
/** app.wxss **/
@import "common.wxss";
.middle-p {
    padding:15px;
}
```

16.1.3 内联样式

框架组件上支持使用 style、class 属性来控制组件的样式。

微信建议，将静态样式统一写到 class 中。style 则用于接收动态的样式，在运行时会进行解析，所以要避免将静态的样式写进 style 中，以免影响渲染速度。但在实际的项目开发中，将频繁改动的样式写进 class 中也会降低开发速度，不如直接写在 style 属性中。待到项目后期进行代码优化时，再将稳定的样式移进 class 也不迟。示例代码如下：

```
<view style="color:{{color}};" />
```

class 用于指定样式规则，其属性值是样式规则中的类选择器名，即样式类名的集合，样式类名不需要带上“.”，样式类名之间可用空格进行分隔。示例代码如下：

```
<view class="normal_view norma2_view " />
```

16.1.4 样式选择器

目前支持的样式选择器包含以下几种。

- .class：例如 .intro，选择所有拥有 class="intro" 的组件。
- #id：例如 #firstname，选择拥有 id="firstname" 的组件。
- element：例如 view，选择所有 view 组件。
- element, element：例如“view, checkbox”，选择所有文档的 view 组件的 checkbox 组件。
- ::after：例如“view::after”，在 view 组件后面插入内容，一般用于实现装饰线效果和 icon 图标。
- ::before：例如“view::before”，在 view 组件的前面插入内容。

在小程序开发中经常用到的是类选择器“.class”和元素选择器“element”。

16.2　CSS 基础

正如之前提到的，CSS 是一种向用户指定文档应如何呈现的语言。Web 浏览器将 CSS 规则应用于组件以影响它们的显示方式。一个 CSS 规则由以下内容组成。

- 一组属性：属性的值更新了 HTML 中内容的显示方式。比如，如果要让元素的宽度是其父元素的 50%、元素背景变为红色，那么分别需要设置 width:50%、background-color:red。
- 一个选择器：它会选择元素，被选择的一个或 *N* 个元素是属性值要作用的目标对象。比如，将 CSS 规则应用到 HTML 文档中的所有 <p> 元素上，可进行如下声明：

```
p {
    ...
}
```

现在让我们来看一个例子，如下代码所示的是包含了两个规则的非常简单的 CSS 示例：

```
h1 {
    color: blue;
    background-color: yellow;
    border: 1px solid black;
}

p {
    color: red;
}
```

第一条规则从 h1 选择器开始，这意味着它将其属性值应用到 <h1> 元素之上了，它包含了三个属性和属性各自的值（每个 属性 / 值 对称为一个声明）。

- 第一个声明将文本颜色设置为蓝色。
- 第二个声明将背景颜色设置为黄色。
- 第三个声明将标题（h1 是标题元素）边框（border）设置为：1 像素宽、实线（不是虚线、点线等）、颜色为黑色。

第二个规则从 p 选择器开始，这意味着它将其属性值应用到 <p> 元素之上了。它包含一条声明，该声明将字体颜色设置为红色。

那么，CSS 是如何工作的呢？

当浏览器显示文档时，它必须将文档的内容与其样式信息相结合。在此过程中，会分成两个阶段来处理文档，具体如图 16-1 所示。

1）浏览器将 HTML 和 CSS 转化成 DOM（文档对象模型）。DOM 在计算机内存中表示文档。它把文档内容和其样式结合在一起。

2）浏览器显示 DOM 的内容。

微信小程序的视图文档树是在 react 虚拟 DOM 的基础之上构建的，其渲染机制与 Web 页面相同。

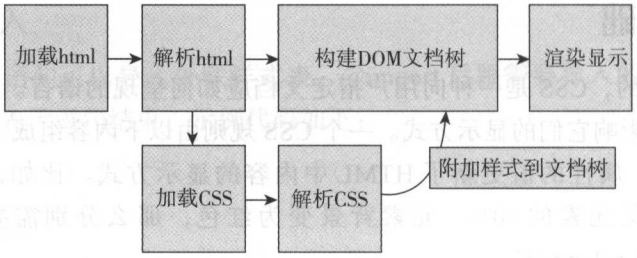


图 16-1

16.2.1 属性与属性值

CSS 是由如下两块内容组合而成的。

- 属性：一些人类可理解的标识符，这些标识符将指出用户想要修改哪些样式特征，例如 font、width、background color 等。
- 属性值：每个指定的属性都需要给定一个值，这个值表示用户想要把那些样式特征修改成什么样，例如，用户想把字体、宽度或背景颜色改成什么样等。

一个属性和其对应的属性值一起组成的属性 / 值对被称作一个 CSS 声明。CSS 声明会被放置在一个 CSS 声明块中。最终，一个 CSS 声明块与选择器相结合形成一个 CSS 规则集。

下面的 h1、p 便是 CSS 规则集：

```
h1 {
    colour: blue;
    background-color: yellow;
    border: 1px solid black;
}

p {
    color: red;
}
```

16.2.2 CSS 声明

为 CSS 属性设置特定的值是 CSS 语言的核心功能。CSS 引擎会通过计算，将对应的 CSS 声明应用到页面的每一个元素之上，从而使得元素能以适当的方式布局，并展示出适当的样式。特别需要注意的是，CSS 的属性和属性值都是区分大小写的。属性和属性值之间，用英文半角冒号（:）分隔，如图 16-2

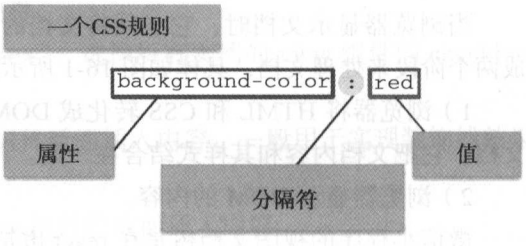


图 16-2

所示。

如果使用了未知属性，或者对属性赋予了无效值，那么该声明会被视为无效，浏览器的 CSS 引擎会完全忽略它。

约有超过 300 个不同的属性以及数倍于属性的属性值。属性和属性值不能任意组合，每个属性都有一个已经定义好的可用属性值范围。在小程序开发中，微信 Web 开发者工具可以给出友好的属性名、属性值的提示，但目前仅限于 WXSS 文件中。

16.2.3　CSS 声明块

声明按块（blocks）分组，每一组声明都用一对大括号括起来，以“{”开始，以“}”结束。

声明块里的每一个声明都必须用半角分号“;”分隔，否则代码不会生效。声明块里的最后一个声明结束的地方，不需要加分号，但是最后加分号是一个好习惯，如图 16-3 所示。

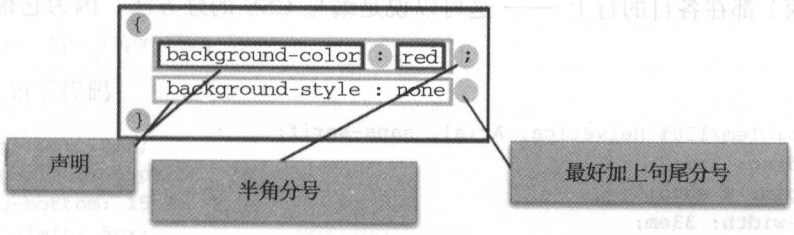


图　16-3

16.2.4　CSS 选择器和规则

如图 16-4 所示，我们需要在每个声明块之前都加上选择器，选择器是一种模式，它能在页面上匹配一些元素。其可使相关的声明仅被应用到被选择的元素之上。选择器加上声明块被称为规则集，通常简称为规则。

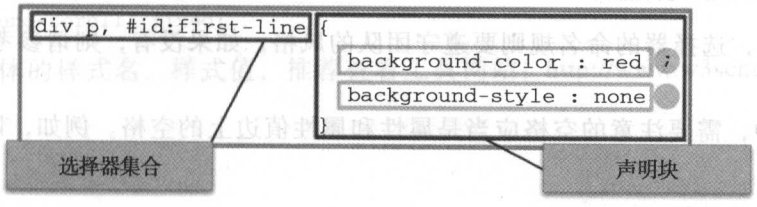


图　16-4

选择器可以很复杂，用户可以制作一个匹配多种元素的规则，这是通过把多个选择器囊括成用逗号分隔的一组选择器来达成的。一个元素可以被多个选择器所匹配，因此，一个给定的属性可能被多个规则设置多次。


注意 如果链或组中的某个选择器无效，比如使用了未知的伪元素或伪类，那么整个选择器组都会无效，这也会导致整个规则全部无效并被忽略。

16.2.5 CSS 最佳实践

CSS 语法并不难写，如果用户写了一些错误的规则，它将仅被忽略。但是，即使有这样的机制，这里也有一些值得了解的最佳实践，它可使 CSS 代码更易于使用和维护。

1. 行间空白

行间空白实际上意味着一些空格、制表符以及一些新行。可以通过添加空格的方式使你的样式表更具可读性。

与处理 HTML 的方式类似，浏览器会倾向于忽略你的 CSS 代码中的许多空格；许多空格的意义仅仅是增加了可读性。在下面的第一个例子中，我们的每一个声明（以及规则的开始 / 结束）都在各自的行上 —— 这可以说是编写 CSS 的好方法，因为它很容易维护和理解：

```
body {
    font: 1em/150% Helvetica, Arial, sans-serif;
    padding: 1em;
    margin: 0 auto;
    max-width: 33em;
}

h1 {
    font-size: 1.5em;
}
```

也可以编写如下所示的样式规则，但这样会更难读：

```
body {font: 1em/150% Helvetica, Arial, sans-serif; padding: 1em; margin: 0 auto;
    max-width: 33em;}
h1 {font-size: 1.5em;}
```

一般来说，选择器的命名规则要遵守团队的风格，如果没有，则请参考 weui 类库的风格。

在 CSS 中，需要注意的空格应当是属性和属性值边上的空格。例如，以下是有效的 CSS：

```
margin: 0 auto;
padding-left: 10px;
```

以下是无效的 CSS：

```
margin: 0auto;
padding- left: 10px;
```

因为 0auto 不被解析为 margin 属性的有效值（0 和 auto 是两个单独的值）。

2. 注释

与其他语言一样，提倡在 CSS 中使用注释，以帮助你或他人在几个月后遇上代码时了解代码是如何工作的，从而理解该代码。

CSS 中的注释以 /* 开始并以 */ 结束，例如：

```
/* Handle basic element styling */
/* ------------------------------------------------------------------------ */
body {font: 1em/150% Helvetica, Arial, sans-serif; padding: 1em; margin: 0 auto;
    max-width: 33em;}
```

3. 速记

诸如 font、background、padding、border 和 margin 的一些属性被称为速记属性，这是因为它们允许用户在一行设置多个属性。

例如，如下这行代码：

```
padding: 10px 15px 15px 5px;
```

等同于如下代码：

```
padding-top: 10px;
padding-right: 15px;
padding-bottom: 15px;
padding-left: 5px;
```

而如下这行代码：

```
background: red url(bg-graphic.png) 10px 10px repeat-x fixed;
```

则与以下这些代码做了相同的事：

```
background-color: red;
background-image: url(bg-graphic.png);
background-position: 10px 10px;
background-repeat: repeat-x;
background-scroll: fixed;
```

关于具体的样式名、样式值，推荐查看工具网站：http://www.w3school.com.cn/html/html_css.asp。

Chapter 17 第 17 章

Go 语言基础

Go 是 Google 开发的一种静态强类型、编译型、并发型，并具有垃圾回收功能的编程语言。Go 语言代码文件以“.go”作为扩展名。如图 17-1 所示，作为高级编程语言进化的末端，它综合了多种语言的优秀特点。

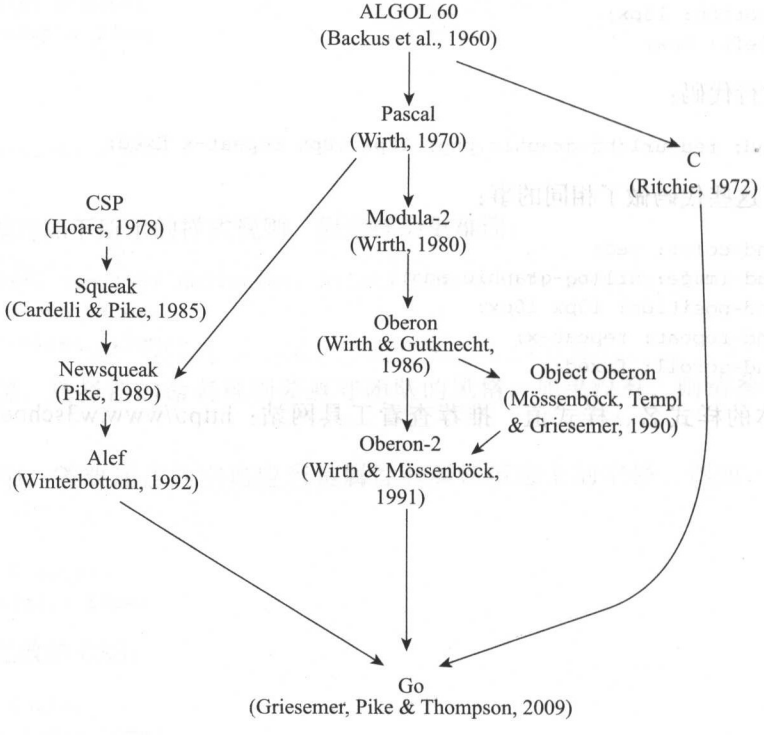


图 17-1

本章将通过一些短小精悍的代码练习，完成对 Go 语言的基础学习。每一节对应一个知识点。本章共 60 余个代码清单，可供读者练习使用。

17.1　基础概念

关键字是编程语言保留使用的，也称保留字。如下代码所示的是 Go 语言所有的保留字。在对变量或者函数命名时，是不能使用这些保留字的。

```
break default func  interface select
case  defer go  map struct
chan  else  goto  package switch
const fallthrough if  range type
continue  for import  return  var
```

相比于其他语言，Go 语言只有 25 个保留字，但其实现的功能却不逊于其他任何一门语言。

17.1.1　hello world 与 import

凡介绍编程语言的书，都免不了要奉上“hello world”这个经典的案例程序，这里也不例外，如下代码所示为最简单的 Go 程序：

```
package main

import "fmt"

func main() {
    fmt.Println("hello, world")
}
```

将之保存为“hello.go”，在命令行终端中执行“go run hello.go”，即可测试代码（前提是已经安装了 Go 语言的开发环境，安装方法参见本书的 7.1 节）。

在上述代码中，第一行代码用于声明包，关于包的讲解请参见 17.1.2 节。

17.1.2　包

每个 Go 程序都是由包组成的。在如下代码中，程序运行的入口即为包 main：

```
package main

import (
    "fmt"
    "math"
)

func main() {
```

```
    fmt.Println(math.Pi)
}
```

按照惯例，包名要与导入路径的最后一个目录一致。例如 math/rand 包就是由 package rand 语句开始的。

这个代码用圆括号组合了导入，这就是“打包”导入语句。在导入了一个包之后，就可以用其导出的名称来调用它。在 Go 语言中，只有首字母大写的名称才是被导出的。

在代码中，可以编写多个导入语句，示例如下：

```
import "fmt"
import "math"
```

不过使用打包的导入语句是更好的形式。

如果包名是目录，那么程序就会去相应的目录中查找，否则就会去 $GOPATH 或 $GOROOT 下查找。

关于使用 import 关键字引入包，有以下几点值得注意。这些技巧虽然不一定都会用到，但了解它们，有助于理解开源项目的源码，尤其是第 3 个技巧，在与数据库驱动相关的项目中经常被使用。

1. 使用点操作符引入包

使用点操作符引入包之后，可以省略包前缀，如下面的代码所示，可以直接使用 Println 方法，代替 fmt.Println：

```
import(
    . "fmt"
)
...
Println("hello world")
```

2. 别名引入

如果同时引入了两个名称相同的包，为了避免麻烦，一般在引入时对其中一个包设置一个别名，如下代码所示：

```
import(
    f "fmt"
)
f.Println("hello world")
```

3. 忽略符“_”

由于 Go 在引入包时，会自动调用包的 init 方法。使用忽略操作符“_”，可以忽略这种自动调用，如下面的代码所示：

```
package sim

import (
```

```
    _ "github.com/go-sql-driver/mysql"
    _ "github.com/mattn/go-sqlite3"
    "github.com/go-xorm/core"
    "github.com/go-xorm/xorm"
)
```

上面的示例源码，摘自 sim.go 类库，地址如下：

https://github.com/rixingyike/sim.go/blob/master/lib/db.go

17.1.3　函数

如下代码是一个函数声明的示例：

```
package main

import "fmt"
func add(x int, y int) int {
      return x + y
}
func main() {
      fmt.Println(add(42, 13))
}
```

在上述代码中，add 与 main 均是函数。函数可以没有参数也可以接收多个参数。在这个例子中，add 接受了两个 int 类型的参数。注意，类型在变量名之后。

1. 多值返回

如果两个或多个连续的函数命名参数都是同一类型时，那么除了最后一个参数类型之外，前面的其他参数类型都可以省略。

在如下代码中，第一行的声明可以简写为第二行的形式：

```
func abs(x int, y int)
func abs(x, y int) //简写
```

函数可以返回任意数量的返回值。在如下代码中，swap 函数返回了两个字符串：

```
package main
import "fmt"

func swap(x, y string) (string, string) {
      return y, x
}
func main() {
      a, b := swap("hello", "world")
      fmt.Println(a, b)
}
```

2. 命名返回值

Go 的返回值可以被命名，并且可以像变量那样使用。返回值的名称应当具有一定的意

义，在使用 gofmt 格式化时，可以作为文档使用。没有参数的 return 语句将返回结果的当前值，即直接返回。在如下代码的方法 split 中，x、y 即是命名的返回值：

```
package main
import "fmt"

func split(sum int) (x, y int) {
      x = sum * 4 / 9
      y = sum - x
      return
}
func main() {
      fmt.Println(split(17))
}
```

函数 split 的 return 即为直接返回。

17.1.4 变量

在下面的代码中，var 语句定义了一个变量的列表。与函数的参数列表一样，类型是放在后面的。var 语句可以定义在包或函数级别，例如变量 c、python 等：

```
package main
import "fmt"

var c, python, java bool

func main() {
    var i int
    fmt.Println(i, c, python, java)
}
```

1. 初始化变量

变量定义可以包含初始值，每个变量对应一个。如果初始化是使用表达式，则可以省略类型；变量可从初始值中获得类型。在如下代码里，对变量 c、python、java 的声明省略了类型：

```
package main
import "fmt"

var i, j int = 1, 2

func main() {
    var c, python, java = true, false, "no!"
    fmt.Println(i, j, c, python, java)
}
```

上述代码在声明变量 c、python、java 时没有指定类型，Go 从字面值猜到了变量的类型。

2. 变量的短声明

在函数中，“:=”用于简洁赋值语句，在明确类型的地方可以用于替代 var 定义。但函数外的每个语句都必须以关键字开始，“:=”结构不能使用在函数之外。在如下代码中，变量 k 使用的即是短声明。一般情况下，仅在 if 语句和 for 语句中使用短声明。

```
package main
import "fmt"

func main() {
    var i, j int = 1, 2
    k := 3
    c, python, java := true, false, "no!"

    fmt.Println(i, j, k, c, python, java)
}
```

17.1.5　基本类型

以下代码所示的是 Go 语言的基本类型：

```
bool 布尔值
string 字符串
int  int8  int16  int32  int64 整型
uint uint8 uint16 uint32 uint64 uintptr 正数
byte uint8的别名，字节
rune int32的别名，代表一个Unicode码
float32 float64 浮点数
complex64 complex128 复数，数学专业会用到，一般开发中极少用到
```

如下示例代码演示了不同类型的变量。同时，与导入语句一样，变量的定义被 var 打包在了一个语法块中：

```
package main
import (
    "fmt"
    "math/cmplx"
)

var (
    ToBe   bool       = false
    MaxInt uint64     = 1<<64 - 1
    z      complex128 = cmplx.Sqrt(-5 + 12i)
)
func main() {
      const f = "%T(%v)\n"
      fmt.Printf(f, ToBe, ToBe)
      fmt.Printf(f, MaxInt, MaxInt)
      fmt.Printf(f, z, z)
}
```

1. 零值

若变量在定义时没有明确的初始化，则赋值为零值。不同数据类型的零值都有其确切的定义，具体如下。

- 数值类型为 0。
- 布尔类型为 false。
- 字符串为 ""（空字符串）。

以下代码所示的是四种基本类型的零值的测试代码：

```
package main
import "fmt"

func main() {
      var i int
      var f float64
      var b bool
      var s string
      fmt.Printf("%v %v %v %q\n", i, f, b, s)
}
```

2. 类型转换

表达式 T(v) 可将值 v 转换为类型 T。以下代码所示的是一些关于数值的转换：

```
var i int = 42
var f float64 = float64(i)
var u uint = uint(f)
```

以下代码所示的是一种更加简单的编写形式：

```
i := 42
f := float64(i)
u := uint(f)
```

在旧版本里，Go 在使用不同类型的变量对新变量进行赋值时需要显式转换，但如今已经不再需要了，以下所示的是测试代码：

```
package main

import (
    "fmt"
    "math"
)
func main() {
    var x, y int = 3, 4
    var f float64 = math.Sqrt(float64(x*x + y*y))
    var z int = int(f)
    fmt.Println(x, y, z)
}
```

可以看到，移除变量 f 的类型声明和显式转换（粗体部分），对结果没有影响。读者在阅读某些 github 项目源码的时候，可能会看到类似的显式转换代码，可将其忽略。

3. 类型推导

在定义一个变量但不指定其类型时，使用没有类型的 var 语句或“:=”语句均可，变量的类型由右边的字面值推导即可得出。

在如下代码中，当字面值常量定义了类型时，新变量的类型与其相同：

```
var i int
j := i // j 也是一个 int
```

但是当右边包含了未指名类型的数字常量时，新的变量就可能是 int 、float64 或 complex128。这取决于常量的精度，如以下代码所示：

```
i := 42           // int
f := 3.142        // float64
g := 0.867 + 0.5i // complex128
```

下面尝试修改以下代码中变量 v 的初始值，并观察控制台打印结果的变化：

```
package main
import "fmt"

func main() {
    v := 42 // 修改这个数值
    fmt.Printf("v is of type %T\n", v)
}
```

17.1.6　常量

常量的定义与变量类似，只不过它使用的是 const 关键字。常量可以是字符、字符串、布尔或数字类型的值，但不能使用“:=”语法来定义。以下所示的是测试代码：

```
package main
import "fmt"

const Pi = 3.14

func main() {
    const World = "世界"
    fmt.Println("Hello", World)
    fmt.Println("Happy", Pi, "Day")
    const Truth = true
    fmt.Println("Go rules?", Truth)
}
```

在上述代码中，Pi、World 均是常量，它们之间仅作用域不同。

关于数值常量

数值常量是高精度的值。一个未指定类型的常量将由上下文来决定其类型。在以下代码中，常量 Big、Small 是由函数来决定其精度的。

```
package main
import "fmt"

const (
    Big   = 1 << 100
    Small = Big >> 99
)
func needInt(x int) int { return x*10 + 1 }
func needFloat(x float64) float64 {
    return x * 0.1
}
func main() {
    fmt.Println(needInt(Small))
    fmt.Println(needFloat(Small))
    fmt.Println(needFloat(Big))
}
```

17.2 条件控制语法

本节主要介绍 for、if、switch 等条件关键字。

17.2.1 for 循环

Go 只有一种循环结构，即 for 循环，基本的 for 循环不使用“( )”，这一点与 JS 语言不同。示例代码如下：

```
package main
import "fmt"

func main() {
    sum := 0
    for i := 0; i < 10; i++ {
            sum += i
    }
    fmt.Println(sum)
}
```

与 C 或者 Java 一样，在 Go 语言中，可以让前置、后置语句为空，示例代码如下：

```
for ; sum < 1000; {
    sum += sum
}
```

甚至可以省略分号，C 的 while 在 Go 中称为 for，示例代码如下：

```
for sum < 1000 {
    sum += sum
}
```

事实上，在 Go 语言里，如果省略了循环条件，那么循环就不会结束了。这种情况适用于 socket 轮询等场景。示例代码如下：

```
func main() {
    for {
    }
}
```

17.2.2　if 语句

与 for 相同，if 语句没有“()”，但“{}”是必需的。示例代码如下：

```
package main
import (
    "fmt"
    "math"
)

func sqrt(x float64) string {
     if x < 0 {
             return sqrt(-x) + "i"
    }
    return fmt.Sprint(math.Sqrt(x))
}
func main() {
     fmt.Println(sqrt(2), sqrt(-4))
}
```

关于 if 的便捷语句

与 for 一样，if 语句可以在条件之前执行一个简单的语句，但是这个语句定义的变量的作用域仅在 if 范围之内，这就是 if 的便捷语句，示例代码如下：

```
package main
import (
    "fmt"
    "math"
)

func pow(x, n, lim float64) float64 {
    if v := math.Pow(x, n); v < lim {
            return v
    }
    return lim
}
```

```
func main() {
    fmt.Println(
        pow(3, 2, 10),
        pow(3, 3, 20),
    )
}
```

在 if 的便捷语句中定义的变量同样可以在任何对应的 else 块中使用，示例代码如下：

```
if v := math.Pow(x, n); v < lim {
    return v
} else {
    fmt.Printf("%g >= %g\n", v, lim)
}
```

17.2.3 switch 语句

除非以 fallthrough 语句结束，否则分支会自动终止，这点与 JS 语言不同，需要显式调用 break，如以下代码所示：

```
package main

import (
    "fmt"
    "runtime"
)
func main() {
    fmt.Print("Go runs on ")
    switch os := runtime.GOOS; os {
    case "darwin":
            fmt.Println("OS X.")
    case "linux":
            fmt.Println("Linux.")
    default:
            // freebsd, openbsd,
            // plan9, windows...
            fmt.Printf("%s.", os)
    }
}
```

switch 的条件会按从上到下的顺序执行，当匹配成功的时候就停止，如以下代码所示：

```
switch i {
case 0:
case f():
}
```

在上述代码中，当 i 等于 0 时不会调用 f。

没有条件的 switch 同 switch true 一样，这一构造使得可以采用更清晰的形式编写长的 if-else 语句，如以下代码所示：

```
func main() {
    t := time.Now()
    switch {
    case t.Hour() < 12:
            fmt.Println("Good morning!")
    case t.Hour() < 17:
            fmt.Println("Good afternoon.")
    default:
            fmt.Println("Good evening.")
    }
}
```

以上代码，switch 逻辑仅会执行一遍。

17.2.4　defer

defer 语句会延迟函数的执行，直到上层函数返回为止。延迟调用的参数会立刻生成，但是在上层函数返回之前，函数都不会被调用，如以下代码所示：

```
package main
import "fmt"

func main() {
      defer fmt.Println("world")

      fmt.Println("hello")
}
```

想象一下，以上程序会先打印哪个单词?

defer 相当于预约了一个延迟调用，它一般用于文件操作句柄等资源的内存释放。延迟的函数调用被压入一个栈中。当函数返回时，会按照后进先出的顺序调用被延迟的函数调用。

17.3　复杂类型

17.3.1　指针

指针保存了变量的内存地址。类型 *T 是指向类型 T 的值的指针，其零值是 nil。如以下代码所示，p 就是一个指针：

```
var p *int
```

在以下代码中，& 符号会生成一个指向其作用对象的指针：

```
i := 42
p = &i
```

在以下代码中，* 符号表示指针指向的是底层的值：

```
fmt.Println(*p) // 通过指针 p 读取 i
*p = 21         // 通过指针 p 设置 i
```

与 C 语言不同，Go 没有指针运算。

17.3.2 结构体

一个结构体 struct 就是一些字段的集合。type 的含义与其字面意思相符，相当于定义了一组字段。在如下代码中，Vertex 是一个结构体，结构体字段可使用点号来访问：

```
package main
import "fmt"

type Vertex struct {
      X int
      Y int
}
func main() {
      fmt.Println(Vertex{1, 2})
}
```

在如下代码中，结构体字段可以通过结构体指针来访问：

```
v := Vertex{1, 2}
p := &v
p.X = 1e9
```

关于结构体声明文法

结构体声明文法表示以结构体字段的值作为列表来新分配一个结构体，使用“Name:”语法仅会列出部分字段，此时与字段名的顺序无关。在如下代码中，Vertex{X: 1} 便指定了 X 字段：

```
v1 = Vertex{1, 2}  // 类型为 Vertex
v2 = Vertex{X: 1}  // Y:0 被省略
v3 = Vertex{}      // X:0 和 Y:0
p  = &Vertex{1, 2} // 类型为 *Vertex
```

其中，特殊的前缀 & 将返回一个指向结构体的指针。

17.3.3 数组

类型 [n]T 是一个包括 *n* 个类型为 T 的值的数组。在如下代码中，声明了一个数组，a 是一个有 10 个整数的数组：

```
var a [10]int
```

在下面的代码中，数组的长度是其类型的一部分，不能改变：

```
package main
import "fmt"

func main() {
      var a [2]string
      a[0] = "Hello"
      a[1] = "World"
      fmt.Println(a[0], a[1])
      fmt.Println(a)
}
```

17.3.4　切片

一个切片（slice）会指向一个序列的值，并且其包含了长度的信息。[]T 是一个元素类型为 T 的 slice。在如下代码中，p 是一个 slice：

```
package main
import "fmt"

func main() {
      p := []int{2, 3, 5, 7, 11, 13}
      fmt.Println("p ==", p)
      for i := 0; i < len(p); i++ {
          fmt.Printf("p[%d] == %d\n", i, p[i])
      }
}
```

Go 语言的数组不能改变大小，但因为有 slice 的存在，不能改变大小的弊端也就不存在了。

1. 重新切片

slice 可以重新切片，创建一个新的 slice 值将指向相同的数组。因此表达式“s[lo:hi]”表示从 lo 到 hi-1 的 slice 元素，包含两端。而“s[lo:lo]”是空的，但“s[lo:lo+1]”则表示有一个元素了。

2. 如何构造 slice

slice 由函数 make 创建，其会分配一个零长度的数组，并且返回一个 slice 指向这个数组。在如下代码中，make 创建了数组 a，该数组拥有 5 个元素：

```
a := make([]int, 5)  // len(a)=5
```

为了指定容量，可将第三个参数传递到 make，示例代码如下：

```
b := make([]int, 0, 5) // len(b)=0, cap(b)=5
b = b[:cap(b)] // len(b)=5, cap(b)=5
b = b[1:]      // len(b)=4, cap(b)=4
```

以下是关于函数 make 的练习代码：

```
package main
import "fmt"

func main() {
    a := make([]int, 5)
    printSlice("a", a)
    b := make([]int, 0, 5)
    printSlice("b", b)
    c := b[:2]
    printSlice("c", c)
    d := c[2:5]
    printSlice("d", d)
}
func printSlice(s string, x []int) {
    fmt.Printf("%s len=%d cap=%d %v\n",s, len(x), cap(x), x)
}
```

3. slice 的零值

slice 的零值是 nil，一个 nil 的 slice 的长度和容量是 0。

4. 如何向 slice 添加元素

向 slice 添加元素是一种常见的操作，在下面的代码中，Go 提供了一个内建函数 append，使用该函数可以向 slice 推入一个或多个元素，并返回新的 slice：

```
func append(s []T, vs ...T) []T
```

append 的第一个参数 s 是一个类型为 T 的数组，其余类型为 T 的值将会添加到 slice。append 的结果是一个包含原 slice 所有元素加上新添加元素的 slice。

如果 s 的底层数组太小，不能容纳所有值时，会分配一个更大的数组。返回的 slice 会指向这个新分配的数组。数组不能改变大小，它仅是一个瓶子，如果瓶子小了，可以换一个大的，要变换的只是指向数组的指针，不变的是数组元素的指针。

17.3.5 range

for 循环的 range 格式可以对 slice 或 map 进行迭代循环。以下是相应的练习代码：

```
package main
import "fmt"

var pow = []int{1, 2, 4, 8, 16, 32, 64, 128}

func main() {
    for i, v := range pow {
        fmt.Printf("2**%d = %d\n", i, v)
    }
}
```

在如下代码中，可以通过使用空白忽略符“_”来忽略序号和值：

```
for _, value := range pow {
    fmt.Printf("%d\n", value)
}
```

17.3.6　map

map 会将键映射到值。map 在使用之前，必须用 make 而不是 new 来创建。值为 nil 的 map 是空的，在 make 之前不能赋值。实例化数组、字典都是使用 make，实例化结构体使用的是 new。

下面是本小节的练习代码，其中 m 即 map 类型：

```
package main
import "fmt"
type Vertex struct {
    Lat, Long float64
}

var m map[string]Vertex
func main() {
    m = make(map[string]Vertex)
    m["Bell Labs"] = Vertex{
        40.68433, -74.39967,
    }
    fmt.Println(m["Bell Labs"])
}
```

在以下代码中，声明 map 的文法跟结构体声明文法相似，不过必须要有键名。其中，Bell Labs、Google 即是键名：

```
var m = map[string]Vertex{
    "Bell Labs": Vertex{
        40.68433, -74.39967,
    },
    "Google": Vertex{
        37.42202, -122.08408,
    },
}
```

关于如何修改 map

- 使用“m[key] = elem”可在 map 类型 m 中插入或修改一个元素。
- 使用“elem = m[key]”可获得元素。
- 使用“delete(m, key)”可删除元素。

在下面的代码中，可通过双赋值检测某个键的存在。如果 ok 为 true，则说明存在键值。

```
elem, ok = m[key]
```

如果 key 在 m 中，则 ok 为 true；否则，ok 为 false，并且 elem 是 map 元素类型的零值。同样，当从 map 中读取某个不存在的键时，结果就是 map 元素类型的零值。

以下是本小节的练习代码：

```
package main
import "fmt"
func main() {
    m := make(map[string]int)

    m["Answer"] = 42
    fmt.Println("The value:", m["Answer"])

    m["Answer"] = 48
    fmt.Println("The value:", m["Answer"])

    delete(m, "Answer")
    fmt.Println("The value:", m["Answer"])

    v, ok := m["Answer"]
    fmt.Println("The value:", v, "Present?", ok)
}
```

17.3.7 闭包

函数也是值，也可以在函数中声明和传递。在下面的代码中，hypot 就是一个函数变量：

```
package main
import (
    "fmt"
    "math"
)

func main() {
    hypot := func(x, y float64) float64 {
        return math.Sqrt(x*x + y*y)
    }
    fmt.Println(hypot(3, 4))
}
```

其中，hypot 就是一个声明为值的函数。

Go 函数可以是闭包的。闭包是一个函数值，它来自于对函数体外部的变量引用。函数可以对这个引用值进行访问和赋值。换句话说，这个函数被“绑定”在这个变量上。

在下面的代码中，函数 adder 返回了一个闭包，每个闭包都被绑定到了其各自的 sum 变量上。函数 adder 返回的不仅是一个做加法运算的函数，还包括上下文环境中的变量 sum，它们作为一个整体被称为函数的闭包。

```
package main
import "fmt"

func adder() func(int) int {
      sum := 0
      return func(x int) int {
            sum += x
            return sum
    }
}

func main() {
      pos, neg := adder(), adder()
      for i := 0; i < 10; i++ {
        fmt.Println(
              pos(i),
              neg(-2*i),
        )
    }
}
```

17.4　方法和接口

本节将学习如何定义方法和接口。方法和接口用于定义对象的行为。

17.4.1　方法

Go 没有类，只能在结构体类型上定义方法。方法接收者出现在 func 关键字和方法名之间的参数中，示例代码如下：

```
func (v *Vertex) Abs() float64 {
    return math.Sqrt(v.X*v.X + v.Y*v.Y)
}
```

其中，“v *Vertex”中的 v 即为方法接收者，*Vertex 是接收者的类型。

可以对包中的任意类型定义任意方法，而不仅仅只是针对结构体。但是不能对来自其他包的类型或基础类型定义方法。在如下代码中，函数“Abs”是定义在 MyFloat 上的方法。MyFloat 是 float64 的别名类型。

```
type MyFloat float64

func (f MyFloat) Abs() float64 {
    if f < 0 {
        return float64(-f)
    }
    return float64(f)
}
```

上面两段代码中定义的 Abs 方法，一个是在 *Vertex 指针类型上，另一个是在 MyFloat 值类型之上。有如下两个原因需要使用指针类型的接收者。

- 避免在每个方法调用中复制值。
- 方法可以修改接收者指向的值。

在下面的代码里，在方法 Scale 中，接收者 v 的成员变量 X、Y 将被改变。

```
func (v *Vertex) Scale(f float64) {
    v.X = v.X * f
    v.Y = v.Y * f
}
```

17.4.2 接口

接口类型是由一组方法定义的集合。接口类型的值可以存放实现这些方法的任何值。在如下代码中，类型 MyFloat、Vertex 均实现了 Abser 接口。在 Go 语言中，一个类型实现一个接口时不需要事先声明，实现本身即是声明。

```
type Abser interface {
    Abs() float64
}
func (f MyFloat) Abs() float64 {
    if f < 0 {
        return float64(-f)
    }
    return float64(f)
}
func (v Vertex) Abs() float64 {
    return math.Sqrt(v.X*v.X + v.Y*v.Y)
}
```

关于 Stringers 接口

如下代码所示的是普遍存在的在 fmt 包中定义的 Stringer 接口：

```
type Stringer interface {
    String() string
}
```

Stringer 是一个可以用字符串来描述自己的类型。fmt 包使用这个接口的 String 方法来进行输出。在如下代码中，fmt.Println 方法在无形中调用了 Person 的 String 方法：

```
type Person struct {
    Name string
    Age  int
}
func (p Person) String() string {
    return fmt.Sprintf("%v (%v years)", p.Name, p.Age)
}
```

```
func main() {
    a := Person{"Arthur Dent", 42}
    fmt.Println(a)
}
```

17.4.3　错误

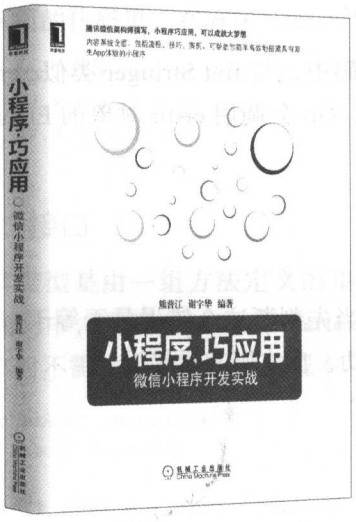
Go 程序使用 error 值来表示错误状态。在如下代码中，与 fmt.Stringer 类似，error 类型也是一个内建接口。在使用 fmt 包打印 error 对象时，Go 会调用 error 对象的 Error 方法获取打印信息：

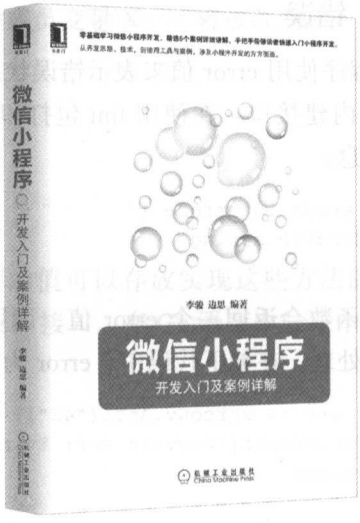
```
type error interface {
    Error() string
}
```

通常函数会返回一个 error 值，调用它的代码应当先判断这个错误是否等于 nil，然后进行错误处理。在如下代码中，error 为 nil 时表示成功，反之则表示错误：

```
if i, err := strconv.Atoi("42"); err != nil {
    fmt.Printf("convert err: %v\n", err)
    return
}
```

推荐阅读





小程序，巧应用：微信小程序开发实战

书号：978-7-111-55682-4 作者：熊普江 谢宇华 定价：59.00元

腾讯微信架构师撰写，小程序巧应用，实现创业大梦想
内容系统全面，包括流程、技巧、案例，
可帮助你简单高效地搭建具有原生App体验的小程序

微信小程序：开发入门及案例详解

书号：978-7-111-56210-8 作者：李骏 边思 定价：59.00元

零基础学习微信小程序开发，精选5个案例详细讲解，
手把手带领读者快速入门小程序开发
从开发思路、技术，到使用工具与案例，涉及小程序开发的方方面面

作者简介

石桥码农，本名李艺，日行一刻创始人，知乎微信小程序专栏作者。国内早期闪客，Adobe 中国首批社区管理员 15 位之一，北京协同创新研究院、中国传媒大学脑科学与智能传媒研究院智能传媒技术中心工程师。个人微信公众号“艺述思维”，致力于将艺术与人工智能相结合，敬请关注。

封面设计：陈潇然

本书作者是难得的文艺型互联网专家，将对艺术的实践和理解同技术相结合，体现出来的是非同一般的技术思维。 微信小程序开发，从提出概念开始，就备受关注。微信作为中国智能手机用户的强依赖应用，以此来打造新的应用平台，形成新的生态圈，既是本身发展的需要，也是开发者和用户的共同需求。小程序的推出，对开发者而言，无疑带来了新的成长机会和岗位需求。小程序开发本身并不高深，但是内容略显庞杂，有一本简明、实操性强的书作为指导，定会事半功倍。本书结构清晰，笔触细腻。和其他“拷贝官方文档”的教程不同，本书案例部分以用 Go 语言为主，综合介绍其他相关知识，显现了本书的原创性。案例翔实精要，体现了作者用心雕琢之功。

——《.NET 安全揭秘》作者 杨文海

小程序带来了全新的机会和平台，本书从生产实践中总结项目必备技能，深入浅出地讲解了小程序的开发技巧，并结合 Go 语言等知识，为读者提供了开箱即用的饕餮大餐。本书不仅有小程序的诸多技巧，更包含了前后端的整合开发，是不可多得的全栈开发指导书。感谢石桥码农，在本书中围绕小程序开发，为读者提供了丰富的知识。

——陈文，上海能伍网络科技有限公司 CTO

互联网产品的推广成本一直是业界关注的焦点，微信小程序因无需安装、触手可及、用完即走等特点，获得了很多企业的青睐和用户的喜爱，但目前除了官方的文档外，少有系统地、完善地讲述小程序开发的图书，本书从 0 到 1、从前端到后端，很好地将微信小程序由浅入深地做了系统讲解，更为难得的是还提供了丰富的实例，非常适合不同层次的读者阅读。本书的面世，对想学习微信小程序开发的工程师来说，无疑是一大福音。

——《深入 Android 应用开发：核心技术解析与最佳实践》作者 苗忠良

上架指导：计算机/程序设计
ISBN 978-7-111-58404-9
9 787111 584049
定价：59.00元

投稿热线：(010) 88379604
客服热线：(010) 88379426 88361066
购书热线：(010) 68326294 88379649 68995259

华章网站：www.hzbook.com
网上购书：www.china-pub.com
数字阅读：www.hzmedia.com.cn